

Title: Group Report

Group 6

Client: Matt Morgan

Supervisor: Louise Knight

Members:

Abdu Dihan (c1301922)

Andrei Hodorog (c1332008)

Ben Leonard-Legarde (c1322567)

Karl Latham (c1335031)

Rhian Milcoy (c1305879)

Robert Horton (c1323141)

Roberto Dyke (c1317850)

Sam Maltby (c1334821)

Zain Tahir (c1308094)

Table of Contents

[Table of Contents](#)

[Introduction](#)

[Progression/Justifications](#)

[Solution Structure](#)

[Acceptance criteria](#)

[Use Cases](#)

[Use case details](#)

[USE CASE 1 - Log into Friendship Application](#)

[USE CASE 2 - Specify which sites are used to extract personal data.](#)

[USE CASE 3 - Find Friendship Strengths](#)

[USE CASE 4 - Find specific friendship strengths](#)

[USE CASE 5 - Results](#)

[USE CASE 6 - Fix Bugs](#)

[USE CASE 7 - Monitor Consumer Base](#)

[Functionalities of Software](#)

[Justification of Scope](#)

[Inter-relationship between system components](#)

[Calculating Friendship Method](#)

[Algorithm development](#)

[Algorithm Justification](#)

[Gathering Data](#)

[Hardware](#)

[Justification of Hardware](#)

[Legal, Ethical and professional issues](#)

[Legal](#)

[Ethical](#)

[Professional Issues](#)

[User Interface](#)

[HCI](#)

[Application Screens](#)

[Basic application flow](#)

[Screen Constraints](#)

[Login \(1\) Screen](#)

[Permissions \(3\) Screen](#)

[Options \(4\) Screen](#)

[Results \(6\) Screen](#)

[Data frames \(Ultra low fidelity Interface design\)](#)

[Wireframes \(Low fidelity Interface design\)](#)

[High Fidelity Design](#)

[Core Algorithms](#)

[Basic Computational Flow](#)

[Data Flow Diagram](#)

[System Class Diagram](#)

[Prototyping](#)

[Facebook API](#)

[TF-IDF Algorithm](#)

[LIWC Algorithm](#)

[Example:](#)

[Friendship Algorithm](#)

[Mathematical Calculation](#)

[Data Handling](#)

[Low Level Data Dictionary](#)

[Facebook API](#)

[Loggedin.php](#)

[TF-IDF](#)

[LIWC](#)

[Code Quality](#)

[Programming Practices](#)

[Language Selection](#)

[Testing](#)

[System Testing](#)

[User's Perspective](#)

[Developer's Perspective](#)

[API](#)

[API-to-Algorithm Conversion](#)

[Algorithm](#)

[Display Information](#)

[Code Quality Testing](#)

[Test Summary report](#)

Work Plan

Risk Analysis

Gantt chart

RASCI chart

Implementation tasks

Critical Path Analysis (CPA)

Diagram

Tabulated CPA

References

Research in requirements presentation

Design report

Introduction

This report marks the end of our significant design work in our development process for this project. This report shall identify our plans for the next stages of the project, comprising of content made since our previous presentation, namely: the solution's structure, the initial user interface design, the core algorithms used, the quality of the code, the testing strategy, our work plan, and a greater insight into the legal, ethical, and professional issues that arise with our proposals. Some of the information that we presented in week 7 shall be included in this report for referential purposes - attendance of the presentation should not be required to be able to understand this document. In this document we identify what we plan on building, we hope this will help align the expectations of our client with our own vision.

Transparency in parts (e.g. the core algorithms section), is intended to provide a means of gaining a greater understanding of our solution from a technical perspective. In addition to code, we have included diagrams and descriptive text, to provide greater accessibility to some of the more complex concepts introduced. As well as providing a wide range of components of our project, we hope to include a great depth of information into them.

In our presentation, we discussed all the necessary research material used to produce our algorithm for calculating friendship. Because this has already been covered, and for the sake of this document retaining a sensible size, we shall not be covering that material again. We do however continue on from our research; we describe our planned implementation of the theories and hypotheses made, and we also attempt to justify these proposals.

Since the presentation that identified the criteria needed to meet our requirements, we have been refining the details. We feel that we have successfully created a platform for us to meet the criteria, and consequently the requirements specified. This is by doing the necessary research in advance, detailing our planned solution (to come), and including a test plan (also to come further in the document). We have not made any changes to the requirements first set, the acceptance criteria specified shall be included in the Solution Structure section of this document for relevant ease of access.

Since the last presentation with our client, we have moved forward quite a bit in the project; some alterations to the initial proposals have been made. We feel strongly that it is imperative for our (previously mentioned) "must have" MoSCoW requirements and most of our "should have" requirements to be met. We also feel that all aspects of the system must function exactly as they should, and all parts must function to as high a standard as each other. The major constraint that we will have to contend with is time. Functionality of the key aspects of the system must come before the quantity of features within the system. Some features and aspects could also be overlooked or over complicated in the design or possibly not work as well in practice as it appeared to on paper.

While we have been designing a solution to our problem, we have been continuously analysing ways in which we can improve our original solution to our problem, this hasn't led to any changes in the original proposal, but I do expect this kind of learning to continue happening over the next semester, so the included prototype code may not be the same as the - justified - the code implemented in the end.

The information included in this report has been selected based on it's the significance to the client we felt it had. In the final meeting of 2014 with our client we discussed whether they'd like to see any additional content added, they decided that the information described in the specification was already appropriate.

Progression/Justifications

The project is at the design stage, we have discussed and settled, on the client's requirements. Once this was achieved we moved onto developing documentation that would help us to produce a quality piece of software. Use cases were developed to ensure the functional requirements were recorded and to model the goals of system/actor (user) interactions; this will insure that programming teams knows what they are developing exactly and who it's going to be used by. Next we produced mock-ups; this was done because stakeholders can interact and understand prototypes more easily than written lists or drawings, and it allows team members to communicate ideas clearly. The programming team researched and built prototypes to ensure the API was fully functional, and that we could get the data that was required for the most important feature of our project: the algorithm to calculate friendship. Using previous knowledge of statistics, combined with research into methods for statistical analysis, and various discussions with PhD students, the friendship algorithm has been iteratively refined to produce a more accurate, representative result. In this report the algorithm specified - we feel - may be attainable with constraints relating to this project (i.e. time, skill, etc.). Testing for the implementation stage was also developed; it shall point out the defects and errors that were made during the implementation phases; it's also essential in ensuring the quality of the product, which in turn ensures the clients requirements are met, and their satisfaction in the application.

We meet regularly with our client. This means we're developing software and adding at least one feature that we demoed or otherwise explained to our client. The meetings have been valuable as the clients feedback ensures that we meet the requirements and produce software that is true to what we set out to produce.

Since the presentation, we have held 2 more client meetings. These were basically to ensure that the project was on the right stage, and that the project was to the standard which the client desired.

Solution Structure

Acceptance criteria

For our presentation we deliberated what our acceptance criteria should be, here they are again in a bullet-pointed format:

Features for the user:

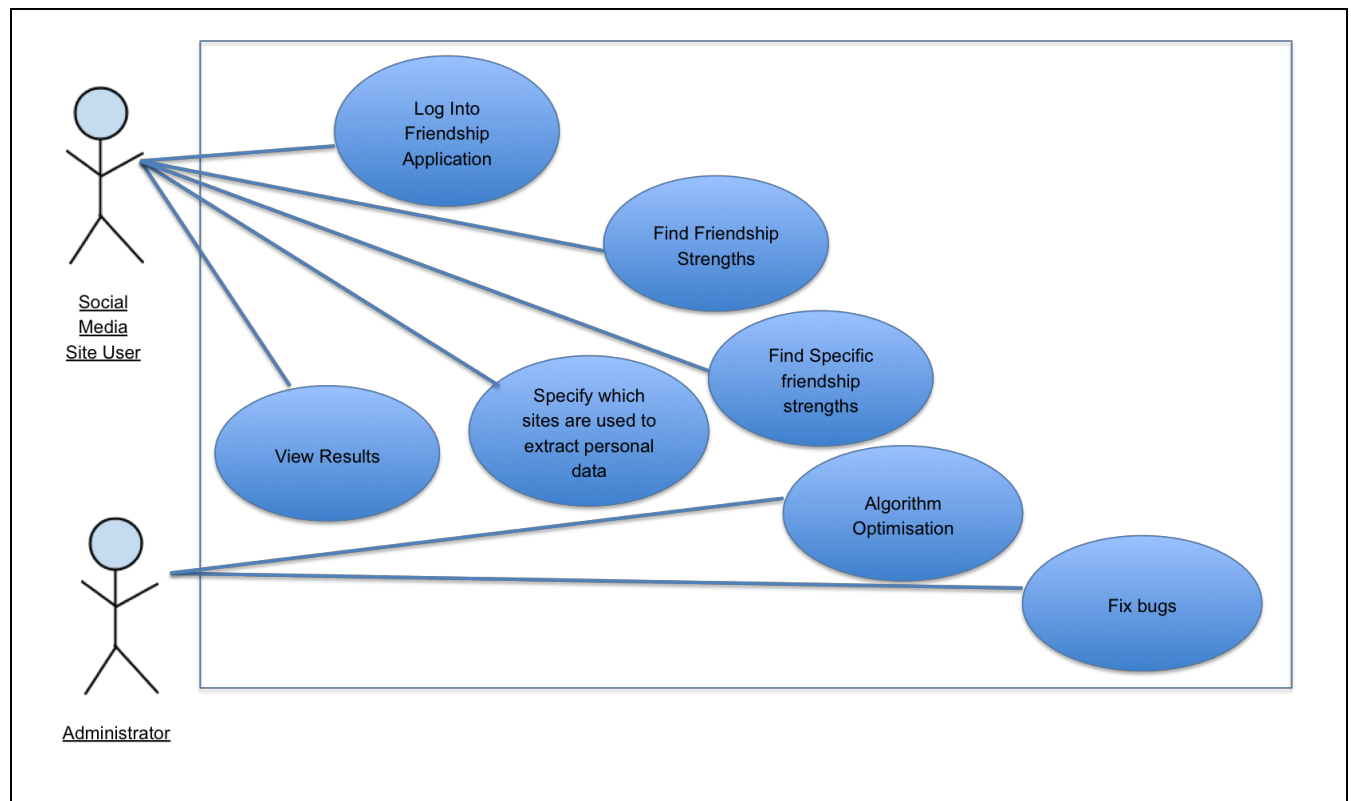
1. See who the closest friends are
2. See the strength between specific friends
3. See number of connections
4. Choose social media sites used & grant system access/permission
5. Have ability to change results
6. Share results with friends
7. See source of greatest interaction with friends

Features for the admin:

1. Fix bugs
2. Add features
3. Monitor consumer base

Use Cases

From the acceptance criteria (above), we created a use case diagram (below) - identifying the actors, and their associated tasks.



Use case details

USE CASE 1 - Log into Friendship Application

Actor:

User/ Customer

Preconditions:

- Must have the correct specifications to run application.
- User must have an account.

Description:

This use case will grant the user access to the application. Also needed for authentication to make sure the application extracts the correct information to find the friendship strength. Also needed to ensure other use cases are carried out correctly.

Basic Flow:1.1 - Enter Username

User proceeds to enter username.

1.2 - Enter Password

User proceeds to enter password.

1.3 - Submit details

User submits details by pressing 'Submit' provided on screen.

1.4 - Confirmation

A message from the system will confirm the user has entered details correctly and will be redirected.

Basic flow completed.

Alternative Flow:2.1 - Incorrect Details (Once steps 1.1 to 1.3 are completed.)

A system notification will appear showing the user that they have incorrectly entered their details, and will give them the option to redo their details.

USE CASE 2 - Specify which sites are used to extract personal data.

Actor:

User / Customer

Preconditions:

- Must fulfil USE CASE 1 preconditions.
- User must be signed in.

Description:

This use case will detail the various websites available to extract data from for the user. Facebook will be the default social media selected. The User can then select various other in addition to Facebook depending on whether they have an account with the associated website.

Basic Flow:1.1 - User selects necessary radio buttons

From the list provided on the screen the user select the websites they would like to analyse.

1.2 - User confirms selections

User submits the websites selected

1.3 - Confirmation Message

System creates confirmation message showing the selected website.

1.4 - Transition / Processing

System now shows the user what strengths are possible to judge how strong a relationship is.

Alternative Flows:

2.1 - Incorrect Websites selected (Steps 1.1 to 1.3 need to be completed)

User decides that selected website are wrong and cancels options.

2.2 - Returns to start of process

User returns to 1.1

USE CASE 3 - Find Friendship Strengths

Actor:

User / Customer

Preconditions:

- Must have completed USE CASE 3
- At least one social media must be selected
- User must have an account with the selected social media

Description:

The user will discover what strengths they have with their entire social circle. This use case will process the instructions detailed in USE CASE 2 and will provide a visual representation with strongest friendships being closest and distant friends being furthest.

Basic Flow:

1.1 - User decides what strengths used

The user will decide what strengths will judge how strong the relationship for their entire list of friends.

1.2 - User submits selected strengths

Once the user has decided what strengths to use, they are given the opportunity to submit their options.

1.3 - Application shows confirmation message

System message appears to detail what strengths have been selected and whether the user is happy with the strengths selected.

1.3 - Systems now processes results

Once the user is happy with the strengths chosen, the application will now start extracting data based on the strengths selected.

Alternative Flows:

2.1 - User re selects strengths (1.1 - 1.3 need to be completed)

If the user was unhappy with the selected strengths they are then given the option to re-select them and return to step 1.1.

USE CASE 4 - Find specific friendship strengths

Actor:

User / Customer

Preconditions:

- Use Case 3 must be completed.

Description:

Once USE CASE 3 is completed the user is then given the option to find out specific information on a certain individual or small group. This will give further details and information regarding the selected friends. It will require further processing.

Basic Flow:1.1 - Select Specific friends to process.

After the results have been displayed for the entire friends list, the user will now have the option to reduce the size of friends they will to analyse.

1.2 - Start analysing / Friends submitted

Once user is happy with selected friends the application will begin the processing again.

Alternative Flows:

N/A

USE CASE 5 - Results**Actor:**

User / Customer

Preconditions:

-Use Case 3 must be completed

-From the social media selected, 'Friends' must be present.

Description:

The results screen presents a visual representation on the friends that are present on the many social media website the user has selected. The user will also have the chance to share, print these results.

Basic Flow:1.1 - Results displayed on screen

After the application has finished the processing, the results will be displayed on the screen.

1.2 - User has the option to share or close application

Once the user has analysed results shown, they have the option to share the results with the social media websites selected.

1.3 - Close Application / Try other social media platforms

Once the user has had the option to share results, the application prompts them to either close the program or return to Use Case 2.

Alternative Flows:

N/A

USE CASE 6 - Fix Bugs**Actor:**

Admin

Preconditions:

- Administrator must have a system account.
- Must have access to all system files

Description:

This use case is purely done on admin side and is not shown to the user at all. Its purpose is to locate bugs to then show to the administrator. This is needed to eliminate as many errors as possible and keep the application stable.

Basic Flow:1.1 - User Bug reports

The administrator analyses reports sent by the user to see whether bugs are present within the application.

1.2 - Admin locates bug

Within the application, the administrator then locates and evaluates the code that the user has identified.

1.3 - Bug fixed

Administrator replaces the code then uploads to create a new application that has eliminated the existing bugs.

Alternative Flows:

N/A

USE CASE 7 - Monitor Consumer Base**Actor:**

Admin

Preconditions:

- Administrator must have a system account.
- Must be a number of users already using the application.

Description:

This use case is required to detail certain information to the administrator. These include web page traffic, the locations in which the traffic is coming from and finally whether users return to discover their social circle again.

Basic Flow:1.1 - Decide what to analyse

Admin decides what to analyse whether its traffic, user details, sales etc.

1.2 - Details of results

Results are shown on screen, with the option to print.

1.3 - Close application

Once admin has finished analysing they then close the application.

Alternative Flows:2.1 - Analyse different detail (1.1 - 1.2 Must be completed)

Once admin has analysed a result, they are then able to return to analyse another specific result.

Return to step 1.1

Functionalities of Software

At the top-level/from the user's perspective of our product we are producing a value for each friendship, identifying tie strength, between the user and their friends. When further analysed, at a lower level this requires us to get the user to log into their social media sites, provide us with permission to analyse their information, then produce accurate, and meaningful results for the user to interpret.

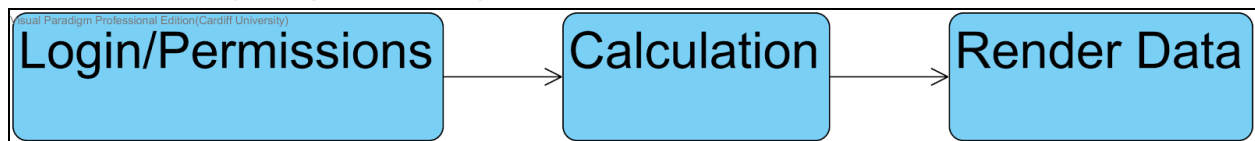
The product shall comprise of two parts, a server-side and client-side part, on the server-side we shall calculate the results for users, and on the client-side we shall attempt to present the results in a meaningful way.

Justification of Scope

Though this may not sound like a lot, this is the crucial part of this application. It is needed to produce accurate and meaningful results, creates a couple problems, namely: how do we accurately calculate friendships, and how do we make them meaningful to the user? Given the complexity of these two problems we felt that it was in our best interest to keep the scope small so we could concentrate on these components, and avoid spreading ourselves too thin.

Inter-relationship between system components

The system comprises of 3 main components, namely: login/permissions, calculation, data rendering. These components make up the basic path required for our program to produce the desired data (as depicted below).



The login/permissions part deals with identifying the user on a social media side so that we may be granted access to their personal information, used to calculate friendship strength; this has a one way relationship with the calculation.

The calculation is a mathematical formula that approximates the user's habits, then uses that information to calculate the strength of ties. This uses numerical information passed through some initial number crunching functions that are tailored to process the data retrieved from the social media sites.

Once the data has been calculated we shall have an interface that displays the data in a way that the user can easily interpret, while presenting as much information as is, appropriately, possible.

Calculating Friendship Method

The problem we were presented with was to, for any individual social media user, provide a map of that user's network of friends and family, taking into account the relative strength of these relationships and ranking them in some form.

The first step towards us completing this was to perform research into what factors affect the strength of a relationship. There were numerous papers detailing various intricacies of this but the one most relevant to us was “Predicting Tie Strength with Social Media” by Eric Gilbert et al. The research showed that the main factors for relationship strength were:

1. Intensity
2. Intimacy
3. Duration
4. Social Distance
5. Reciprocal Services
6. Emotional Support
7. Structural

We decided to focus our app on Facebook, as that is one of the most widely used and therefore can portray a better representation of the relationships. Our app allows people to log in and then through use of the API's we acquire data about their interaction habits with their friends, we then parse this through an algorithm we designed to produce a score for each friend.

The data that we are collecting to calculate the score, in order of importance is:

#	Name	Factor affected
1	Wall words exchanged	Intensity
2	Participant-initiated wall posts	Intensity
3	Friend-initiated wall posts	Intensity
4	Inbox messages exchanged	Intensity
5	Inbox thread depth (number of messages sent between pair)	Intensity
6	Participant's status updates	Intensity
7	Friend's status updates	Intensity
8	Participant's photo comments	Intensity
9	Friend's photo comments	Intensity
10	Participant's number of friends	Intimacy
11	Friend's number of friends	Intimacy
12	Days since last communication	Intimacy

13	Wall intimacy words	Intimacy
14	Inbox intimacy words	Intimacy
15	Appearances together in photo	Intimacy
16	Participant's appearances in photo	Intimacy
17	Distance between hometowns (miles)	Intimacy
18	Distance between Time zones (hours)	Intimacy
19	Friend's relationship status	Intimacy
20	Days since first communication	Duration
21	Links exchanged by wall posts	Reciprocal Services
22	Applications in common	Reciprocal Services
23	Number of mutual friends	Structural
24	Groups in common	Structural
25	Norm. TF-IDF of interests and about	Structural
26	Wall & inbox positive emotion words	Emotional Support
27	Wall & inbox negative emotion words	Emotional Support
28	Age difference	Social Distance
29	Number of occupations difference	Social Distance
30	Educational difference (degrees)	Social Distance
31	Overlapping words in religion	Social Distance
32	Political difference (scale)	Social Distance

Algorithm development

A linear equation is used to calculate the friendship value, producing a continuous value between 0 and 1.

To improve the accuracy of the calculation this algorithm shall take the user's usage into account when calculating friendship, and weigh the results accordingly. We can take this into

account by taking a sample of the friendships, calculating an average figure for each of the different variables used. This way data is only significant if it differs largely from the norm, i.e. if a user does not normally use walls posts but uses them frequently with one of their friends it implies they have a stronger relationship and so it shall be given a bigger weighting

Algorithm Justification

We have decided to create this modular algorithm to calculate tie strength so that alterations may be made to the data put into it to achieve a more accurate result without having to change the whole algorithm. The proposed algorithm still abides to the theory presented in previous research, though our algorithm gets the result in a programmatically friendly manner. Further discussion of the friendship algorithm is discussed in the Core Algorithm section of this document, to a more technical degree.

Gathering Data

In our presentation we briefly covered our analysis of different social media sites we could use to gather the data required for this algorithm. We discovered that some websites facilitated certain friendship variables better than others.

Social Network	Intensity	Intimacy	Duration	Reciprocity	Social Structure	Emotiveness	Social distance
Facebook	5	5	5	5	5	5	5
Twitter	2	2	3	1	2	1	1
LinkedIn	1	1	3	1	5	1	5
Foursquare	1	1	1	1	2	1	1
Pinterest	1	1	1	5	3	1	1
Google Plus	5	5	5	5	5	5	5
Tumblr	1	1	5	5	3	1	1

Rating	Value	Colour
Poor	1	1
	2	2
Ok	3	3
	4	4
Good	5	5

These approximate statistics have helped us in comparing different sites, and prioritising their implementation in our project. Below are the rankings of social media sites - in terms of value gained by implementing them. Since Facebook, and Google Plus have so many features, we also compared the number of users of the respective sites to help us rank them.

Social Media Site Ranking

1. Facebook (1.35 billion users)^[1]
2. Google Plus (359 million users)^[2]
3. Twitter (288 million users)^[3]
4. Tumblr
5. Pinterest

6. Foursquare

This analysis has caused us to prioritise accessing Facebook's API first as it provides a wide range of variables required, and the users of Facebook are the best fit to our social media user population. Once we have implemented the Facebook API in our application, we plan to look towards implementing more social media APIs to get a great range of our potential users.

Hardware

To implement our solution we plan on hosting the system on a public server. In this case, server-side hosting solves some problems we may encounter if we were to make this a client-side application. Client machines shall be required to run the application, and will only work through a web browser, using somewhat modern web rendering tools (e.g. <canvas>) to present our results.

In our presentation we compared different potential server solutions for hosting the proposed system (slide contents below). While developing the system we plan to host it privately on a production server; so we may create code, test it, and see the effect of our changes on performance.

Shared hosting		VPS		Dedicated hosting	
Advantages	Disadvantages	Advantages	Disadvantages	Advantages	Disadvantages
<ul style="list-style-type: none"> - The cheapest solution (~ between 25 and 150 pounds per year) 	<ul style="list-style-type: none"> - Low resources - Cannot handle more than 10k visitors (server requests) / day - Shared resources with other customers - Not suitable for a large scale application 	<ul style="list-style-type: none"> - More resources than a shared hosting solution 	<ul style="list-style-type: none"> - More expensive (~between 150 and 530 pounds per year) - Preinstalled Operating System (usually not able to chose the desired Linux distribution fully compatible with the scripts) - Resources still shared to some extent with other customers - Not suitable for full customization of the resources 	<ul style="list-style-type: none"> - Access to the command line through SSH - Exclusive access to the - Dedicated IP address, suitable for secure communication that could gain trust from the users – e.g. installing an SSL certificate signed by a CA (Certificate Authority) 	<ul style="list-style-type: none"> - The most expensive solution (~between 60 and 280 pounds per month)

Justification of Hardware

Due to the increase in low-powered devices designed for specific use cases (i.e. tablets for web browsing, Chrome Cast for streaming video) we are seeing fewer powerful, general hardware devices (like desktops), which can do many more calculations in a short amount of time. Whilst we push toward ubiquitous computing, we find a higher necessity for centralised computing; thus to allow users to experience our application in a predictable, and unvaried way we must use a server. Server-side processing is ideal as we can estimate the computation times of user requests with greater accuracy as there are fewer variables

involved, i.e. slow internet/ slow processor/ small memory. This allows the system admin to be able to alter the amount of processing time each application instance has to calculate the friendship value for individual users.

Though if it is necessary to change it from a server-side to a client-side application, in the future, we may easily run it on a user's computer using virtualisation, without any radical redesigns, or too much extra programming.

Legal, Ethical and professional issues

Legal

Laws have been put in place that will affect the running of the application. Copyright Law will affect us directly as we will need to ensure we are neither being a target to offenders nor unlawfully using the work of other people. To ensure this, it is important to create a copyright notice for the application. This is needed to show users clearly that the application is subject to copyright, show directly who the copyright owner is, and will deter and 3rd party users to extract data and be a culprit of plagiarism. It is important to include the copyright notice on every page of the application so no loop holes are discovered.

To ensure that we are not breaking copyright law, it is important to show acknowledgments for people's work. If we use people's work it should be expressed clearly making sure it is in clear view. I would suggest the copyright notice for our work should be "© Cardiff University, Group 6 2015 ©".

When we designed the terms and conditions that will be shown at the start of the application, we made sure that all users must abide by these rules with no option to bypass this step. Within the terms and conditions we will include information that will declare use of any potential copyright use (E.g. Profile pictures) and information that will be used based on the target user.

This will then also abide by the data protection Act 1998 which means that when creating the application we will make every effort to create a secure service where data is dealt with efficiently and in a responsible manner as mentioned in the ethical section.

Ethical

When creating this application, we intend to do nothing illegal. The aim of the application is to access and analyse user's personal details which means that there are many ethical issues that surround the subject. We aim to acquire the user's information accurately making sure that we are only taking information that is needed to produce the intended end result. This will be detailed in the consent that the user must agree to. Once the data has been extracted, we will use secure servers and networks to ensure that no third party can access information. No data will be stored for longer than a stated period of time that the user will be aware of.

The application will store basic information about the user. These will include; name, email address and age. When storing this data we will ensure that encryption is used and data is stored in a secure environment. This will affect data protection laws, so we need to ensure that the user has access to this data and have appropriate control over what information is available to the developers. The data should be accurate, adequate, sufficient for its purpose, relevant, and is kept up to date by the user.

When using images in our system, we will make every effort to only use only the profile picture belonging the user from their chosen social media site. This will reduce the risk of using any imagery that is protected by copyright law. When using the profile picture of the user, we will ensure that we are granted permission. If the User declines, there will be a basic stock photo used which will not affect any laws regarding copyright.

Professional Issues

As a group throughout the creation of this application, we have made every effort to act and remain professional. With the current agile development model, it have given us the chance to meet with the client on a regular basis. When meeting the client we make up most effort to ensure that the group are well prepared and present new information in a professional manner.

User Interface

HCI

The way users shall access our application shall be via a web browser, due to the wide range of devices a web browser can be implemented on, and the limited variety of input methods users have on these devices, we must design our application to support the basic functionality; so the application can be used on as many devices as possible. Keyboard inputs, and mouse/pointer actions will be used to manipulate the application via a GUI (Graphical User Interface), this will allow us to supply information to the user in an appealing form; as well as fitting into the standard design of modern web apps the user will be able to use our application more intuitively.

Application Screens

Based on our original analysis of the application use cases, the app will consist of 4 main screens (listed below), with several sub-screens that appear in cases where the program is loading data, or an error occurs. Below is our application's "UI Roadmap" - identifying the fundamental concept of its navigation.

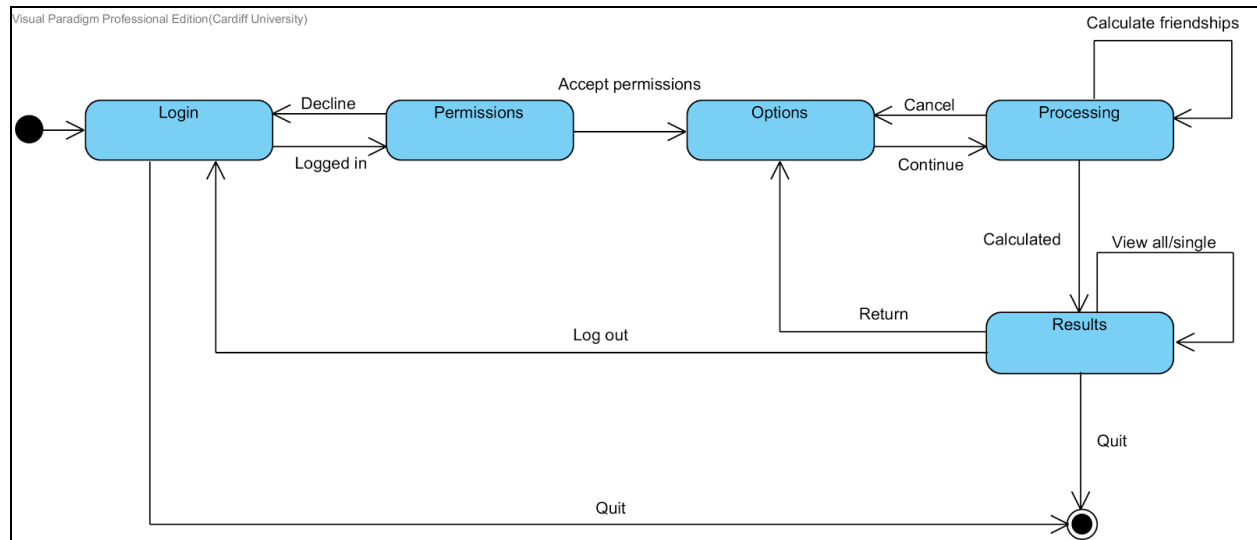
Screens:

- (1) Login
 - (2) Failed to login
- (3) Permissions
- (4) Options

- (6) Results
 - (5) Processing

The flow through the screens above is very linear; the numeric value, in parenthesis, indicates the order the user is likely to see these screens in.

To make it slightly clearer below is a diagram of the individual screens represented as different states, identifying the application's flow.



Basic application flow

The 'login' screen is used to gain access to the social media sites used by our application, this will send the user to a verification screen generated by a social media site. The user shall then be led onto the permission screen, where we shall ask the user for the required permissions to calculate the friendship score. From here the user will select and confirm the result options (what information they wish to view). The program will then go and calculate the result(s) from the user's information, while displaying a processing screen. Once the application has finished calculating the data, the results shall then be displayed on another screen.

Screen Constraints

Login (1) Screen

If the user fails to log into the application they shall be led to the *Failed to login (2)* screen. The screen will allow the user to either attempt to log in again, or exit the application. If the user is already logged in, or succeeds at logging in, they shall be taken to the next screen - the *Permissions (3)* screen.

Permissions (3) Screen

The permissions screen requests the user to grant the application access to specific personal details about themselves. The user can either accept the request, or decline it, if the user

declines the request then they shall return to the original *Login* (1) screen, otherwise the user shall be sent on to the *Options* (4) screen; the user can also exit the application if they choose to.

Options (4) Screen

The user will be able to specify who they want the program to analyse, whether it be just their family members, a few select friends, or everyone. The user may either exit the application, cancel the operation, and return to the *Login* (1) screen, or go to the next screen - Results (6).

Results (6) Screen

While the user is waiting for their results to be calculated the application shall present the *Processing* (5) screen; the user may cancel the program mid-way through *Processing* (5), they shall be led back to the *Login* (1) screen.

Data frames (Ultra low fidelity Interface design)

The diagrams below specify the information each screen contains, and some primitive structure.

(1) Login

Social media site name label	Screen name label (e.g. Login)
Username/email label	Username/email text field
Password label	Password text field
Submit button	

(2) Failed to login

Social media site name label	Screen name label (e.g. Login)
Username/email label	Username/email text field
Password label	Password text field
Submit button	
Error message label	

(3) Permissions

Social media site name label	Screen name label (e.g. Permissions Required)
Permissions required paragraph	

Accept button	Decline button
---------------	----------------

(4) Options

Log Out	Screen name label (e.g. Options)
Analyse everyone radio button	
Analyse individual friendship radio button	Drop down list of friends
Continue button	Cancel button

(5) Processing

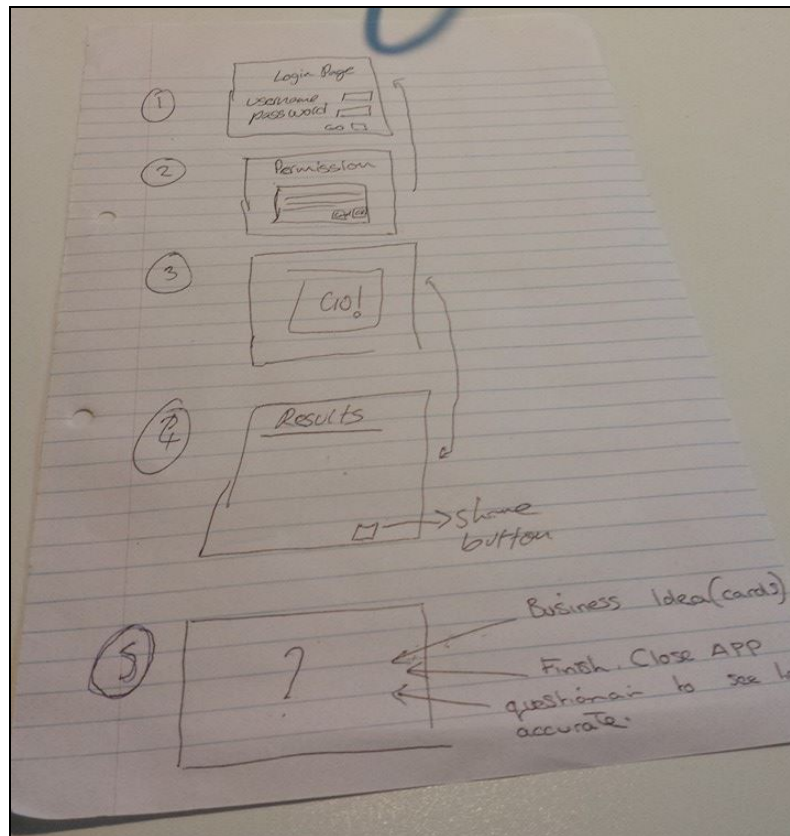
	Screen name label (e.g. Processing)
Progress bar/loading pinwheel	
	Cancel button

(6) Results

Log Out	Screen name label (e.g. Results)
Friendship value(s) view	
	Back

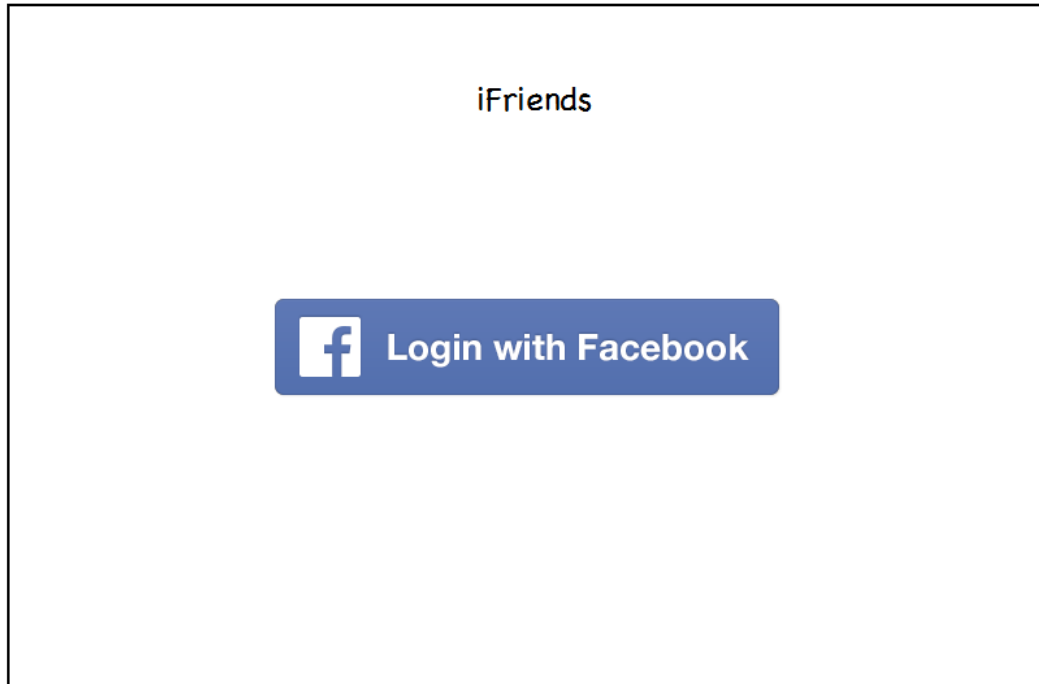
Wireframes (Low fidelity Interface design)

Below are the original drafted wireframes that we presented to our client [\[8\]](#). During the meeting we presented to him what we believed the requirements was and discussed this with him to ensure we were on the same page. We then presented our hand drawn mock-ups to him and got his feedback on it. Using that feedback we altered our design and moved to developing digital wireframes.

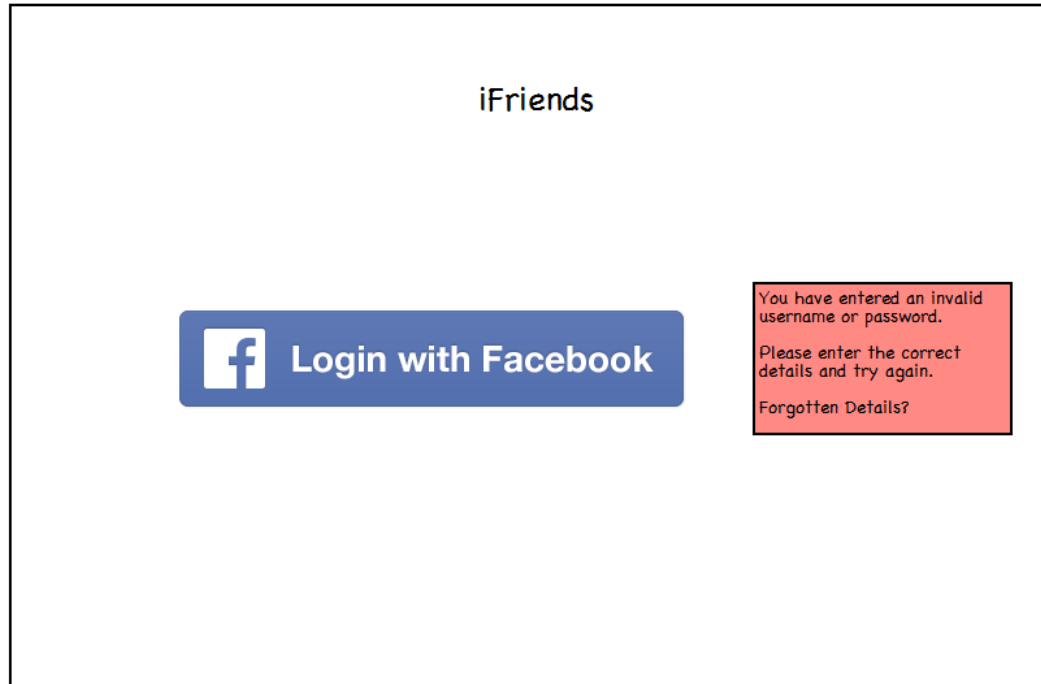


Our wireframes have been evaluated by potential users, and users with technical expertise of what we are trying to do, to make sure our interface makes sense. In the first iteration we made several mistakes, which we have now addressed; problems with missing navigation links, and exiting the application in a secure manner have been addressed thanks to us testing our interface at this stage - preventing potentially major headaches later on. In the proceeding frames we have altered them to address the UI errors.

(1) Login



(2) Failed to login



(3) Permissions



(4) Options

iFriends

Log Out

All ,friends, family or that special one?

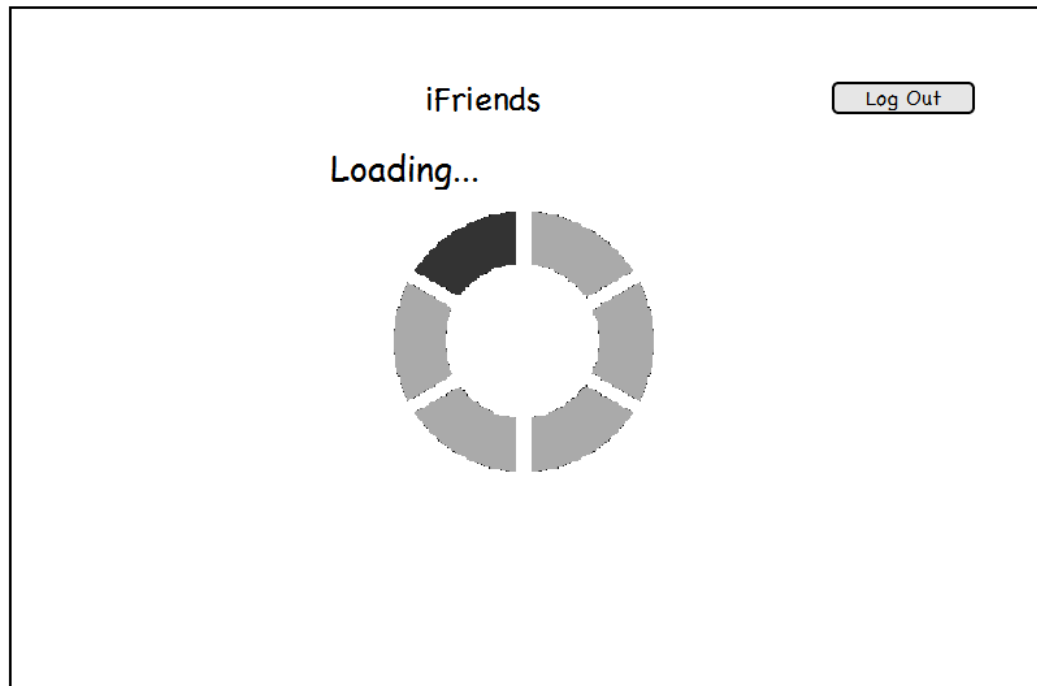
☒ All

☐ Friends only

☐ Family only

☐

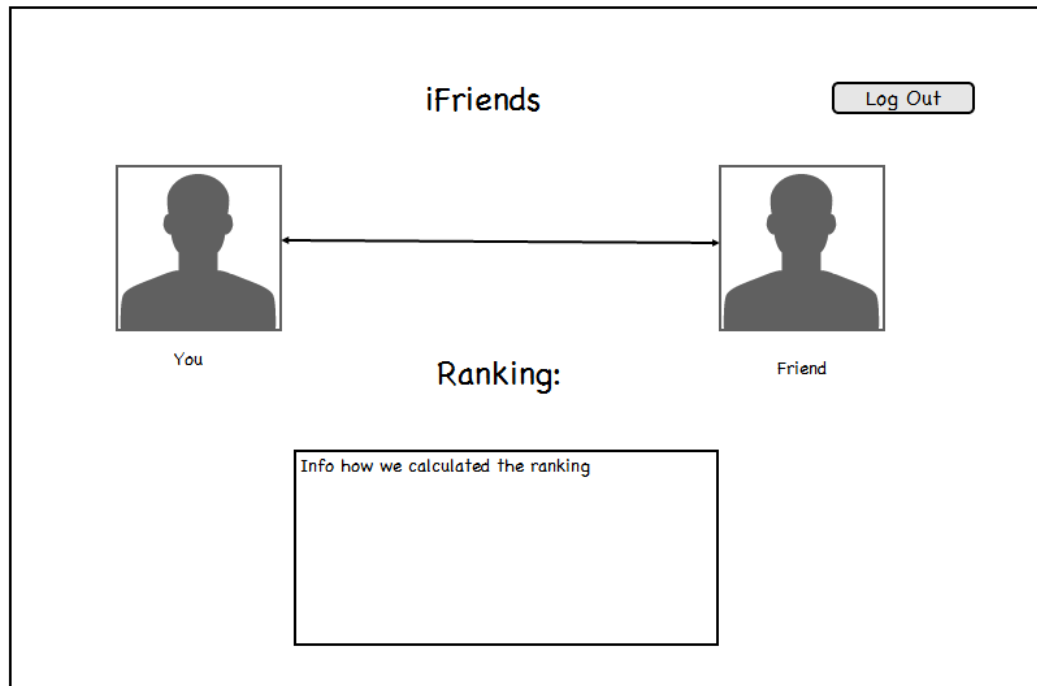
(5) Processing



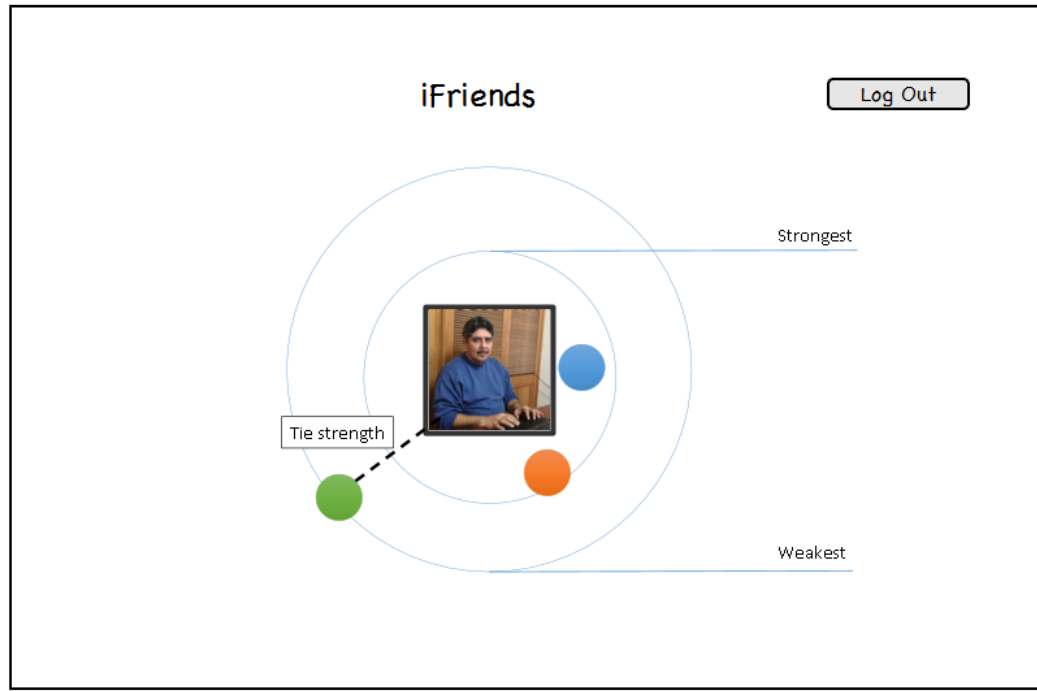
(6) Results

Results may be displayed in two forms: subject-to-friend (single result), and subject-to-friends (all results). The subject-to-friends provides a top-level view of all friends selected by a user, allowing them to easily compare their friendship to others. Subject-to-friend provides the user with a stripped back view of their friendship, reducing the amount of information presented to them, and preventing confusion. By implementing these two views we gain the advantages that both provide, and we can also provide an easy route to implement further calculations, for individual friendships, in the future without having to do more development on the user interface.

(6.1) Results (Subject-to-friend)



(6.2) Results (Subject-to-friends)

**High Fidelity Design**

We plan to do more work on refining the user interface, and by producing these simple, yet descriptive representations we have been able to identify a structure that includes all the required functionality of our program, and we have been able to produce a sensible design layout. These high fidelity designs ought to help reduce any potential confusion caused by the previous low-fi designs.

(1) Login

iFriends



Login with Facebook

(2) Failed to login (Password)

iFriends

 Login with Facebook

Email or Phone

[Keep me logged in](#)

Password

[Forgotten your Password?](#)

Log in

Please re-enter your password
The password you entered is incorrect. Please try again (make sure
your caps lock is off).

Forgot your password? Request a new one.

(2.1) Failed to login (Email Address)

iFriends

 Login with Facebook

Email or Phone

[Keep me logged in](#)

Password

[Forgotten your Password?](#)


Log in

Incorrect email address
The email you entered does not belong to any account.

You can log in using any email address, username or mobile phone number associated with your account. Make sure that it is typed correctly.


(3) Permissions

iFriends

 Request for Permission

iFriends is requesting permission to do the following:

- Access my basic information**
Includes name, profile picture, gender, networks, User ID, list of friends and any other information I've shared with everyone.
- Access my contact information**
Current Address and Mobile Number
- Access messages in my inbox**
Private messages, and Group Messages
- Access posts in my News Feed**
Wall Post, Pictures, Statuses
- Access my Facebook Chat Messenger**



iFriends


Logged in as Joe Bloggs? ([Not you?](#))

Allow

Don't Allow

(4) Options

iFriends


 Log out

All friends, family, or that special someone?

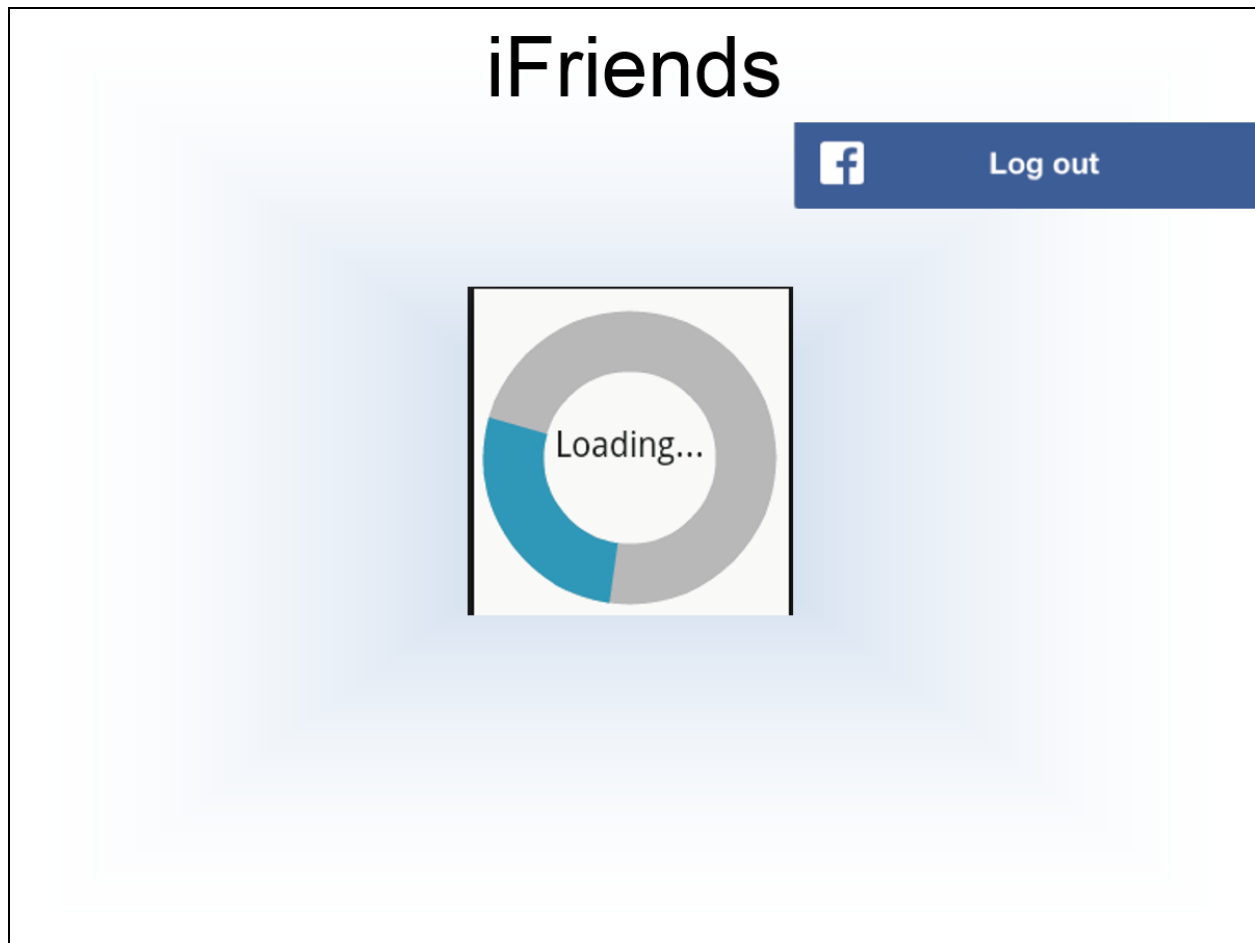
☒ All

☐ Friends only

☐ Family only


☐ 

(5) Processing




(6) Results (Subject-to-friend)


iFriends


 Log out

Joe Bloggs



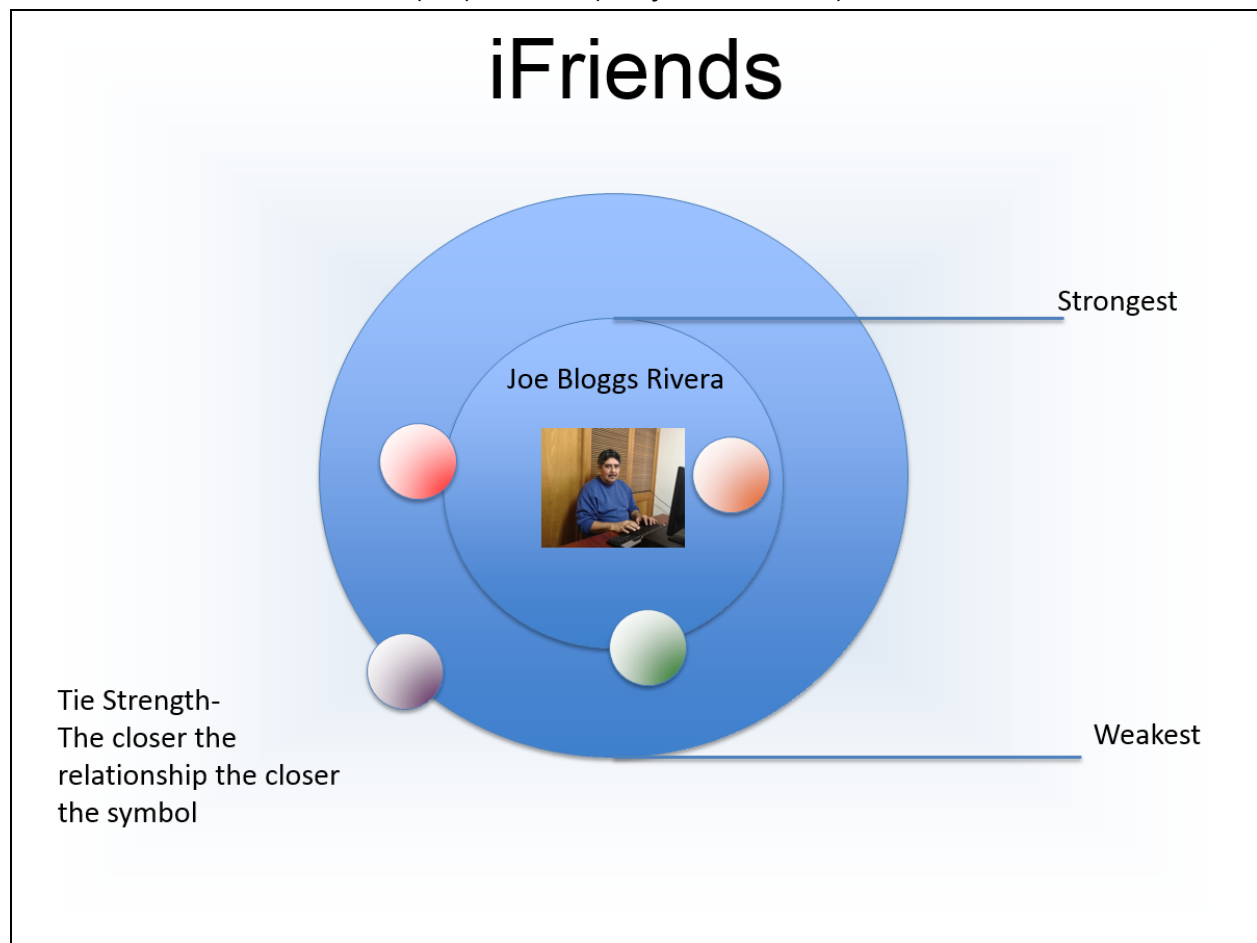
Friend





Ranking:
Information on how we calculated the ranking

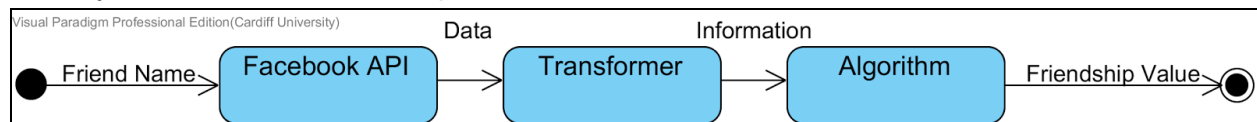
(6.1) Results (Subject-to-friends)



Core Algorithms

Basic Computational Flow

To calculate a friendship value we must take three steps: 1. Get the data about the friendship, 2. transform the data into an appropriate numerical information, and 3. calculate the value from the numerical information. By splitting the process into three parts we are able to create a somewhat modular interface between components, allowing specific parts of the algorithms involved with calculating these results to be edited without affecting the rest of program, since the only interaction between the parts is data.



The Facebook API is one of the social media sites we are planning on retrieving information from, we shall request that part of our program to provide us with the required set of data. This data shall be passed to the transformer code, which takes data and converts it into a

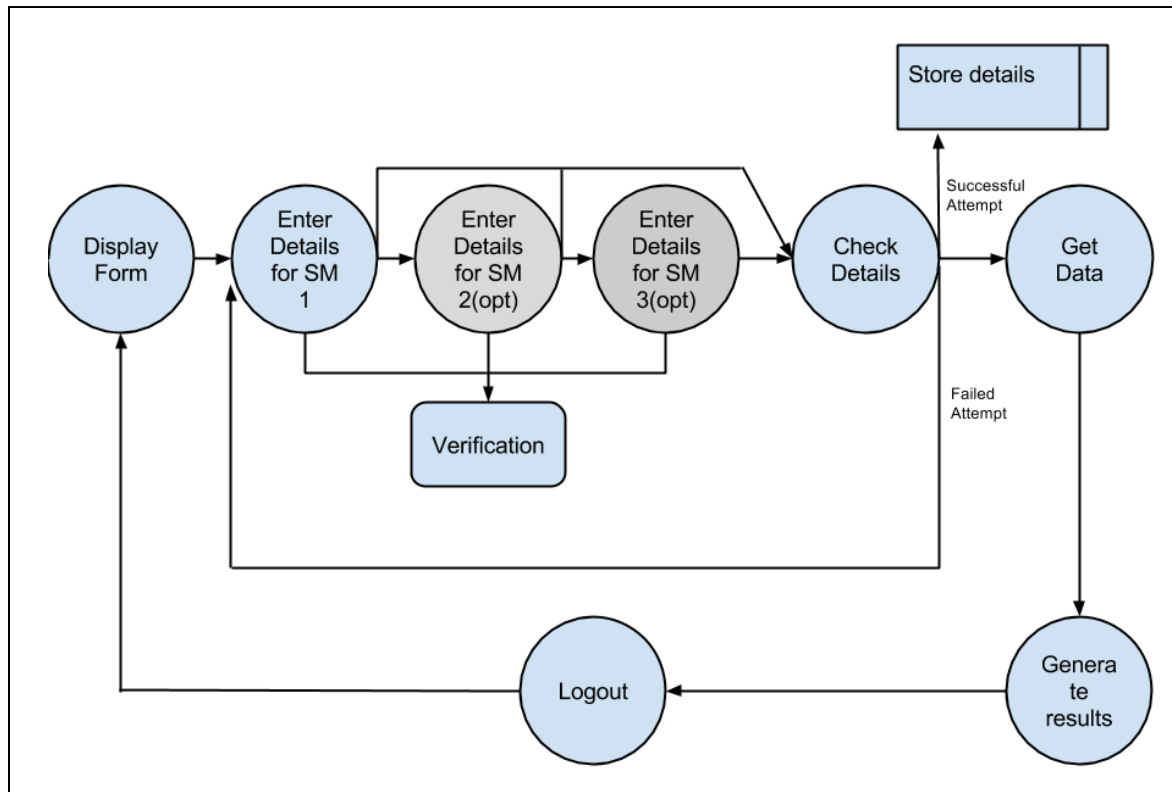
numerical result (*see TF-IDF Algorithm for an example of this type of transformation*). The algorithm is then provided with this information, and calculates the resulting friendship value. By designing the program in this fragmented manner we have been able to specialise our coding team's programming roles. Our team has several people working on getting the information required from the API, we have several people working on transforming that data into something we can process with minimal additional conversion, and we have several people specialising in coding the algorithm. By programming in a non-sporadic manner we can study our own code, allowing us to identify efficiency, and semantic issues in our own code without worrying about anyone else's code.

Data Flow Diagram

The figure below is an abstract-ified diagram of the flow of data in our system, showing what data we get, and where it goes.

Explanation:

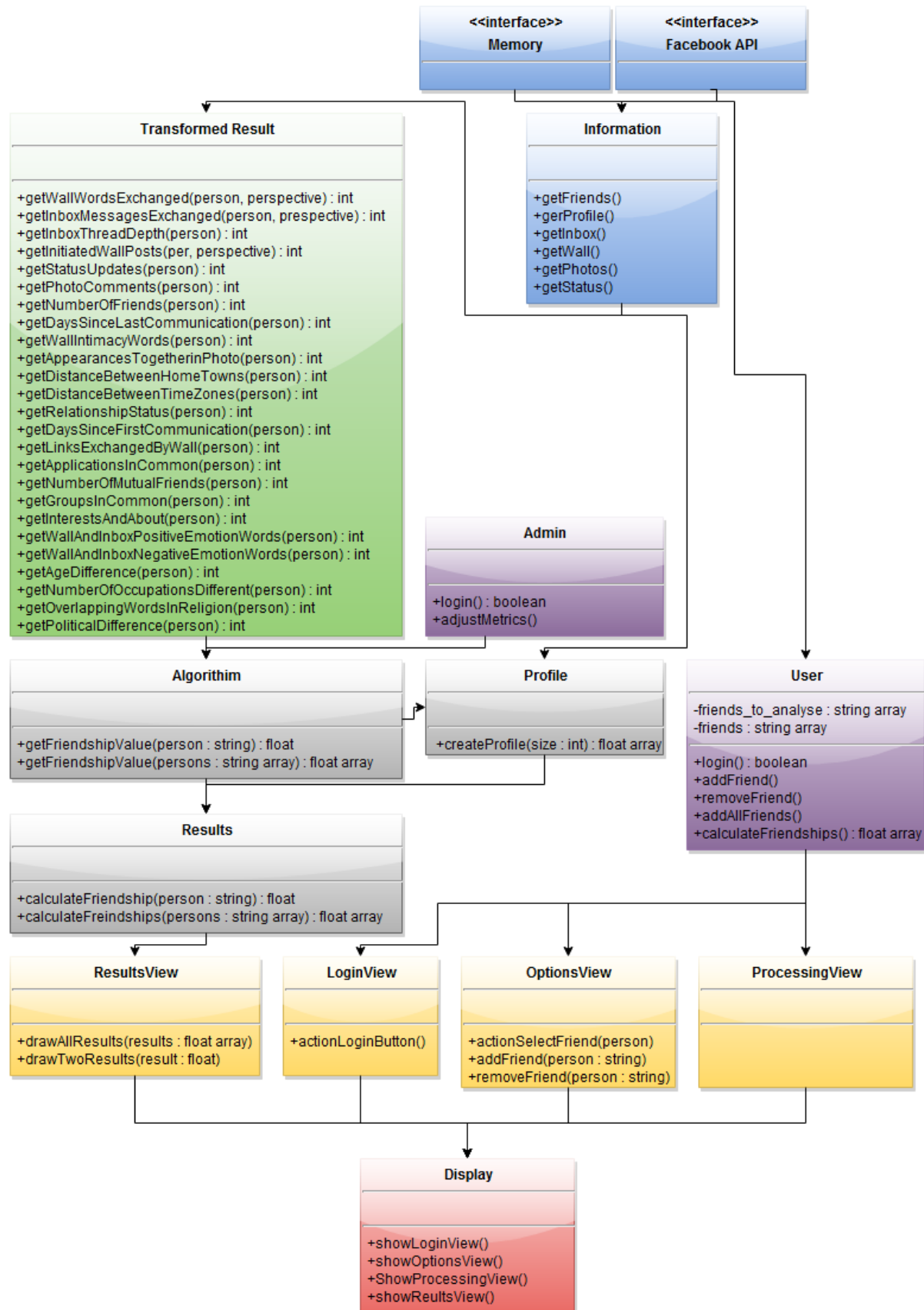
1. One user must enter their details for one social media site.
2. These details are then verified.
3. Details can be entered for 2 more social media sites (optionally)
4. Once the user decides that the details entered matches their requirements, the user submits the details
 - A successful attempt will proceed to get the data from the sites
 - A failed attempt will prompt the user to enter their details again
5. These details will then be used to obtain the required data used to generate the results
6. The results will then be generated



System Class Diagram

The diagram below contains all the important classes, and the methods required to use them, note that the minimalist approach to mutator (A.K.A setter) methods is due the system not requiring us to define much information. At the top of the diagram we find the low-level parts of the system - APIs, raw data, etc. - as we traverse further down the diagram we find we reach the high-level parts, where the user can interact with the system. If you analyse the diagram closely you will notice that the relevance, and clarity of the information in the system improves the further down the diagram we go.

The diagram has been selectively coloured to identify the different functional components of the system. The blue colour indicates the access point of raw data, the green colour indicates the class where we transform this data into useful information, the colour purple identifies the actors of the application, the grey boxes represent classes that are used to calculate the strength of a friendship, the yellow classes are the respective views for the application, and the red view controls each of the yellow classes.



Prototyping

We decided to prototype some features of our program to identify potential implementation problems - allowing us to design our program to accommodate for these problems. It has also helped us gain invaluable experience at developing the solution, whether that be learning PHP, or deepening our understanding of social media APIs.

Facebook API

The Facebook API prototyping was performed to test the capabilities offered by Facebook via their API service and the restrictions they impose on these features.

One of the initial ideas for measuring friendship strengths via the API was to perform a lookup of all the content ever posted by the user going back as far as the user wished. The program would then analyse the names of all their friends who 'liked' the users content and tally who liked the largest number of individual posts in the shortest timespan to create a list of your top ten (data prevailing) Facebook friends.

The second feature implemented in the prototype is a tool designed to page through the entirety of the user's inbox (ignoring message content!) with the intention of parsing the names of the recipient and sender for every message. The tool then uses this information to work out how many messages a user has sent and received from each correspondence. This is used to create another list of your top ten friends albeit very slowly.

This implementation has been written in PHP and JavaScript. I chose to mix the two as I found it much simpler to initiate a Facebook session in JS then 'hand' the active session over to PHP via cookies than to work solely in one language or another.

As this is merely a prototype there is minimal front end with little (if any) styling, the first non-prototype iteration however, will feature full CSS styling with a simplistic and easy to use interface.

TF-IDF Algorithm

Term Frequency Inverse Document Frequency (TF-IDF) is a calculation used to reflect how important a term (or terms) are in relation to a collection of terms. The formula comprises of two parts: term frequency, and inverse document frequency - specified below.

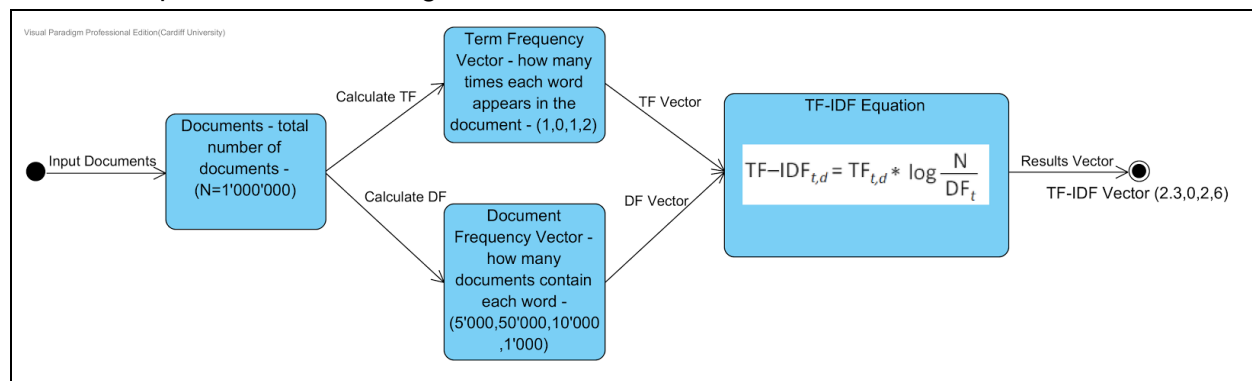
	Description	Calculation
Term Frequency	The number of times each term appears in a <i>single</i> document	-

Inverse Document Frequency	How much information each word provides. I.e. How rare it is across a <i>set</i> of documents	Dividing the total number of documents by the number of documents containing the term and then taking the logarithm of that quotient
----------------------------	---	--

The term frequency is then multiplied by the inverse document frequency; the formula to calculate TF-IDF has been included below:

$$\text{tf-idf}(t,d,D) = \text{tf}(t,d) \times \text{idf}(t,D)$$

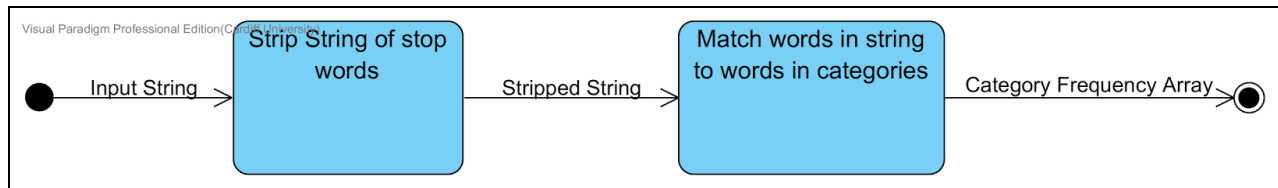
The following diagram shows the transformation of a set of documents, and how they are converted from quantifiable information into numerical values, which may be used in our algorithm. The application of TF-IDF on a set of documents is an excellent example of the kind of complex methods our algorithm needs to handle data transformation.



We are implementing a TF-IDF algorithm to help identify similarities between two people's interest and about pages; this technique was used by a report, which we are basing our data manipulation on. The report uses the information gained from this algorithm to help calculate any structural discrepancy between the user and their friends. By implementing this algorithm in our project we are more likely to attain a result of similar accuracy to the one of the report; you may find the prototype implementation for this algorithm just below, this implementation has been written in PHP. It scans through each word in a set of sentences (In the form of arrays) and calculates the corresponding TF-IDF of each word.

LIWC Algorithm

LIWC is a text analysis algorithm that aims to calculate the degree to which people use different categories of words across a wide array of texts. In our project, we aim to analyse the messages exchanged between people in social media. In our software prototype, we split the words into the following categories: body, family, friends, health, home, leisure, money, religion, sexual, swears & work.



All the words in each category are stored in separate dedicated text files. In each text file, there is one word per line. The PHP implementation prototype takes a sentence as an input, eliminates the stop words (and the words having a length less than three characters) and parses the text files associated with the categories mentioned above (body_sorted.txt, family_sorted.txt, etc.), trying to find matches. The stop words are defined in the stopwords.txt file, one per line. The words found associated with each category are then counted and the output data is returned in JSON format.

Example:

Input sentence: I've had a long day at work - I'm on my way home now. How is the family?

Words found after dropping stop words: long, work, home, family

Categories matched:

- Home – 1 word (word: “home”)
- Family – 2 words (word: “home”, “family”)

JSON Output:

```
{
  "body":0,
  "family":2,
  "friends":0,
  "health":0,
  "home":1,
  "leisure":0,
  "money":0,
  "religion":0,
  "sexual":0,
  "swears":0,
  "work":0
}
```

The JSON format is suitable as an output in the scenario of a web application because it could be easily parsed and plotted into graphs using JavaScript libraries such as D3.js, jqPlot, etc.

All the words extracted from the files are initially stored in arrays. In order to match the words extracted from the input text to different categories in a fast and efficient manner, binary search was used as the search algorithm.

Because we expect large data sets in the production environment, we have to use an efficient algorithm that parses the data quickly, in order not to increase the response time of our application. Binary search has a complexity of $O(\log_2 n)$, compared to linear search, that has a complexity of $O(n)$; the complexity (and implicitly, the processing time) of linear search increases significantly with the array size, compared to binary search, for which the complexity increases much slower.

Binary search implemented in PHP:

```
function binarySearch ($val,$a,$low,$high) {
```

```

    if ($high < $low) {
        return -1;
    }
    $mid = $low + intval(($high-$low)/2);
    if ($a[$mid] > $val) {
        return binarySearch ($val,$a,$low,$mid-1);
    } else if ($a[$mid] < $val) {
        return binarySearch ($val,$a,$mid+1,$high);
    } else {
        return $mid;
    }
}

```

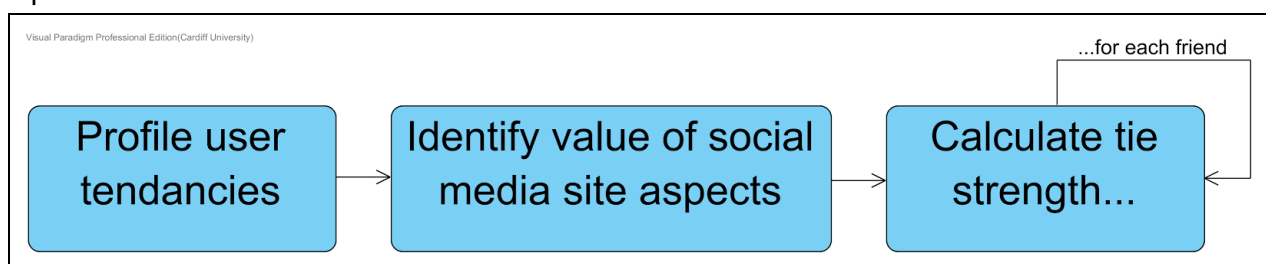
PHP was used as a programming language for the implementation of this prototype because of the fast processing speed in a web application environment. Considering the sensitivity of the data that has to be processed and the length of the data sets expected to be used in the production environment, all the processing needs to be done on the backend side, the data being processed on the server (and not on the client side, as it would have been the case with JavaScript, for example).

The working version of this prototype can be found at the following address:

<https://project.cs.cf.ac.uk/A.Hodorog/words/words.php>

Friendship Algorithm

The 'Friendship Algorithm' is the algorithm we have designed to solve the problem of accurately predicting tie strength between the user and their friends. This algorithm has three stages (can be seen in diagram below), the first stage comprises of profiling the user's usage tendencies on a social media site - specifically analysing how they use a social media site. With this stage complete the algorithm can then identify the key aspects of a user's social media site to analyse, this helps provide greater accuracy to the final result, and helps us optimise performance with a minimal loss in the accuracy of the result. With the key aspects identified, the algorithm may analyse each friendship and assign a value to them based on where they fit with the user's usage profile. This basic flow is depicted below, with a loop on the final state, this is to highlight that the user profiling, and key aspect identification, only have to occur once - per social media site - while the tie strength for multiple friends may be repeated with the same values created.



The generation of a profile of the user's average activities is pivotal for this algorithm. In many machine learning systems the algorithm is passed a training set of data, for the algorithm to

be accurate, the system must be provided with a large amount of data. Our algorithm requires a set of test data to calculate the average case, we do this for individual users, as different users use with social media sites in different ways.

Mathematical Calculation

Once we have generated a profile of the user, and all the individual variables (listed in the table above) are calculated, we can calculate the strength of an individual friendship using the simple mathematical equation below. The value of n is the length of the list of the individual variables calculated for an individual friend. The value of μ is calculated from the profile generated for the user. The value of x is the value found from analysing a friend's information; by subtracting x from μ we can identify the conformity of x . The value of y compensates for any discrepancy found by a result (i.e. μ and x correlate positively ($y:=+1$), or μ and x correlate negatively ($y:=-1$)).

$$\sum_{i=1}^n (x_i - \mu_i) \cdot y_i$$

Where x is an array of the raw data, n is the size/length of x , μ is an array of scalar variables, and y is an array of multiplier variables.

(LaTeX code: `\sum_{i=1}^n (x_{i} - \mu_{i}) \cdot y_{i}`)

The values from this equation are then made proportional, so that they can be compared, then the final value can be produced.

Data Handling

To avoid running into any legal issues, we have decided to not store any information collected from social media sites. The only time data shall be stored (in temporary memory) in our system is when someone is using the application, we plan to drop the data when the user exits the application.

Low Level Data Dictionary

We have produced a low level data dictionary to identify the information we store in the system.

All files mentioned are included in the prototyping folder of the appended zip file.

Facebook API

Entity:	Matlabv3.html (JavaScript)		
Variable Name	Description	Type	Default Value
appld	Facebook Unique App ID	String	"1506880812915540"

XFBML	Use Facebooks' Parser	Bool	TRUE
version	Facebook API version	String	"v2.2"
cookie	Save session as cookie	Bool	TRUE
fjs	Facebook SDK Object	Object	'http://connect.facebook.net/en_US/sdk.js'
username	Store Users Facebook name	String	""
input	Remove login button	Object	

Loggedin.php

Entity:	loggedin.php		
Variable Name	Description	Type	Default Value
\$username_value	POST'd username from previous page	String	""
\$helper	Facebook login object	Object	
\$session	Retrieves active login session	Object	
\$endname	Store Users Facebook name	String	""
\$namelist	Stores final query results	Array	NULL
\$next_page	API QuesryString	String	"/me/feed?fields=likes&limit=25"
\$page_exists	Flags when no more pages are available	Bool	TRUE
\$request	Facebook API request object	Object	
\$response	Executed request response object	Object	
\$graphObject	GraphObject from from \$response	Object	

\$assoc_posts	'data' field as an an array	Array	NULL
\$splode	\$next_page exploded by '/' delimiter	Array	NULL
\$assoc_post	Users posts as an array	Array	NULL
\$post_id	Post ID for current loop	String	""
\$created_time	Post 'created date' attribute for current loop	Object	
\$likes	Post 'likes' attribute for current loop	Object	
\$assoc_likes	Array of \$likes objects	Array	NULL
\$assoc_like	Array of \$assoc_likes JSON strings	Array	NULL
\$alltentemp	Sorted array of 'likers' Facebook names	Array	NULL
\$na	Current users Facebook name	String	""
\$allten3	\$alltentemp with the users name removed	Array	NULL
\$allten	Array of names with occurrence removed	Array	NULL
\$count	Tally array for 'likers' names	Array	NULL
\$hi10flip	\$count array flipped for descending order	Array	NULL
\$sorted	Final short list of top 10 friends	Array	NULL

TF-IDF

Entity:	tf_idf.php		
Variable Name	Description	Type	Default Value
\$bundle	Pre-defined example input array	Array	[1]'We are nine and a half hours ahead of grenwich'...
\$dictionary	Word list created from the input	Dictionary	NULL
\$terms	Word list exploded from the input array	Array	NULL
\$countDoc	Tally array of term occurrence	Array	NULL
\$temp	Temporary storage array	Array	NULL
\$index	Occurrence count	Integer	NULL
\$entry	Index entries	Array	NULL

LIWC

Entity:	liwc.php		
Variable Name	Description	Type	Default Value
\$message	Input message to analyse	String	""
\$val	The value to be searched	String	NULL
\$a	Array to be searched	Array	NULL
\$low	Starting position index	Integer	NULL
\$high	Ending position index	Integer	NULL

\$mid	Middle position index	Integer	NULL
\$file	File name to be opened	String	stopwords.txt
\$stopWords	Processed array containing stop words	Array	NULL
\$handle	Text file object to be parsed	Object	NULL
\$string	String to be stripped of punctuation etc...	String	NULL
\$matchWords	Words that match the stop-words from \$string	Array	NULL
\$wordCountArr	Array counting how often the words appear	Array	NULL
\$wordsArr	Words parsed from \$handle are added here	Array	NULL
\$categories	Listing of all the categorised stopwords	Array	["body"]parse_words_file("body_sorted.txt"); ["family"]....

Code Quality

Programming Practices

Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behaviour. Agile teams^[9] maintain and extend their code a lot from iteration-to-iteration, and without continuous refactoring, this is hard to do.

Continuous Integration is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early. By integrating regularly, you can detect errors quickly, and locate them more easily.

Single coding standard (including everything from tabs vs. spaces and curly bracket placement to naming conventions for things like classes, methods, and interfaces), everything just works better. It's easier to maintain and extend code, to refactor it, and to reconcile integration conflicts, if a common standard is applied consistently throughout.

Language Selection

The decision has been made early on that the best way forward for our application to be web based, for reasons which are outlined in other sections of this report. As such it is clear that web based languages will be essential.

A web based project means a flavour of HTML is essential in order to structure the information shown on the web pages. As a number of different developers will be working on these XHTML would be a logical choice, as this can be validated to ensure its conformity with the start and end tags.

As styling a document within HTML is now largely depreciated, thus CSS will be required. This is a forced decision as there is no other way to conveniently style a document.

Initially we considered doing all of the system processing and validation work entirely server side, but when discussed as a group we felt that when multiple users where making use of the system it would put too great a load on the server. A web based language to support client side validation, and other client side work would be required in order to help reduce this load. JavaScript is cross browser, and cross platform compatible and is therefore a logical choice.

A language is required to access the social media APIs, but also a language which works in a web browser. We gave some consideration to using a Python plugin, as this would also allow the application to work stand-alone from a web browser as part of future development. However as we have decided to go forward with an entirely web based project we have chosen PHP as our language; this also reduces the range of technical expertise required to maintain the system.

Testing

The main idea behind testing is to prove that the final product meets the client's requirements via a series of tests; the tests are usually done in two parts: The tester makes sure that all of the required features and functions are present, and accounted for; this is done via testing from a developers perspective, and then, to make sure that the features and functions behave as expected we have developed tests from the user's perspective. Testing from the user's perspective also gives us a chance to test whether we have met our acceptance criteria - specified in the previous presentation.

System Testing

User's Perspective

User testing is valuable to determine whether the application will be popular and successful. We have decided that a simple table format would best suit the client side testing as it will be easy for the user to follow and enter the results. The results will be anonymised to comply

with laws and regulations. Each user will be given a unique user ID to allow results to be located efficiently.

Note: When testing we assume the user has compatible hardware and software to run the application.

LOGIN SCREEN					
User ID					
Subject	Expected	Actual	Mark / 10 for Expected vs. Actual (10 being Best)	Additional Comments	Time Taken
Login Screen					
Details required					
Feedback throughout process					
Time taken to Complete Login					

Failed To Login Screen					
User ID					
Subject	Expected	Actual	Mark / 10 for Expected vs. Actual (10 being Best)	Additional Comments	Time Taken
Feedback on Login error					
Re-enter of details					

Feedback once successful					
Time Taken to login after error					

Permissions Screen				
User ID				
Subject	Expected	Actual	Mark / 10 for Expected vs. Actual (10 being Best)	Additional Comments
Permissions Asked				

Options Screen				
User ID				
Subject	Expected	Actual	Mark / 10 for Expected vs. Actual (10 being Best)	Additional Comments
Options Available				
Suitable for Task in Hand				

Options available to change presentation				
--	--	--	--	--

Results Screen					
User ID					
Subject	Expected	Actual	Mark / 10 for Expected vs. Actual (10 being Best)	Additional Comments	Time Taken
Expected Layout					
Results Displayed readable?					
Accuracy of Results					
Processing time to calculate results					

Post Results Screen					
User ID					
Subject	Expected	Actual	Mark / 10 for Expected vs. Actual (10 being Best)	Additional Comments	
Options to Share results					

Navigation available after results				
Overall Satisfaction				

Developer's Perspective

Since these tasks are all completed by the server, our environment never changes, so in the interest of conciseness, we shall assume the test environment is the same as specified in the hardware section; other than this change we have stuck to a standard software testing test case design [\[5\]](#).

For information on content of 'Inbox', 'Wall', 'Status', 'Photo', 'Friend', and 'User Profile' look at the data storage specification.

API

Test Case ID	#001
Test Case Summary	To test the API - Get all Inbox messages
Prerequisites	User is logged in, we have permission from user to access Inbox messages
Test Procedure	1.Select the user(me)/friend 2.Systematically request messages 3.Iteratively receive all messages
Test Data	TestUser account
Expected Result	All of the user(me)/friend's messages are retrieved
Actual Result	
Status	
Remarks	

Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	
Date of Execution	

Test Case ID	#002
Test Case Summary	To test the API - Get all Timeline (Wall) posts
Prerequisites	User is logged in, we have permission from user to access Timeline posts
Test Procedure	<ol style="list-style-type: none"> 1.Select the user(me)/friend 2.Systematically request Timeline posts 3.Iteratively receive all Timeline posts
Test Data	TestUser account
Expected Result	All of the user(me)/friend's Timeline posts are retrieved
Actual Result	
Status	
Remarks	
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	

Date of Execution	
-------------------	--

Test Case ID	#003
Test Case Summary	To test the API - Get all Status posts
Prerequisites	User is logged in, we have permission from user to access Status posts
Test Procedure	1.Select the user(me)/friend 2.Systematically request Status posts 3.Iteratively receive all Status posts
Test Data	TestUser account
Expected Result	All of the user(me)/friend's Status posts are retrieved
Actual Result	
Status	
Remarks	
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	
Date of Execution	

Test Case ID	#004
--------------	------

Test Case Summary	To test the API - Get all Photo information
Prerequisites	User is logged in, we have permission from user to access Photo information
Test Procedure	1.Select the user(me)/friend 2.Systematically request Photo information 3.Iteratively receive all Photo information
Test Data	TestUser account
Expected Result	All of the user(me)/friend's Photo information are retrieved
Actual Result	
Status	
Remarks	
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	
Date of Execution	

Test Case ID	#005
Test Case Summary	To test the API - Get all Friend information
Prerequisites	User is logged in, we have permission from user to access Friend information

Test Procedure	1.Select the user(me)/friend 2.Systematically request Friend information 3.Iteratively receive all Friend information
Test Data	TestUser account
Expected Result	All of the user(me)/friend's Friend information are retrieved
Actual Result	
Status	
Remarks	
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	
Date of Execution	

Test Case ID	#006
Test Case Summary	To test the API - Get all User Profile information
Prerequisites	User is logged in, we have permission from user to access User Profile information
Test Procedure	1.Select the user(me)/friend 2.Systematically request User Profile information 3.Iteratively receive all User Profile information
Test Data	TestUser account

Expected Result	All of the user(me)/friend's User Profile information are retrieved
Actual Result	
Status	
Remarks	
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	
Date of Execution	

API-to-Algorithm Conversion

Test Case ID	#007
Test Case Summary	To test the information acquired through the API is correctly converted to a computable form that can be implemented with our Friendship Algorithm.
Prerequisites	We have successfully acquired the correct information through the social media site's API.
Test Procedure	1.Input test data 2.Process test data 3.Output test data
Test Data	See Friendship Algorithm design for specification of data set to pass through
Expected Result	All of the user(me)/friend's User Profile information are retrieved
Actual Result	

Status	
Remarks	
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	
Date of Execution	

Algorithm

Test Case ID	#008
Test Case Summary	Ensure the generated profile of the user is accurate
Prerequisites	We have successfully acquired the correct information through the social media site's API.
Test Procedure	<ol style="list-style-type: none"> 1.Input test data 2.Process test data 3.Output test data
Test Data	See Friendship Algorithm design for specification of data set to pass through.
Expected Result	Result should follow the mathematical equation specified (in the Friendship Algorithm design specification).
Actual Result	
Status	

Remarks	
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	
Date of Execution	

Display Information

Test Case ID	#009
Test Case Summary	Ensure the diagram of the data is accurate to the result calculated by the algorithm
Prerequisites	We have successfully run the algorithm and have accurate results to display
Test Procedure	<ol style="list-style-type: none"> 1.Input test results 2.Draw test results on canvas 3.Display canvas on the user's screen
Test Data	See Friendship Algorithm design for specification of data set to pass through.
Expected Result	Result should accurately depict the results from the Friendship Algorithm
Actual Result	
Status	
Remarks	

Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	
Date of Execution	

Code Quality Testing

To test that the code is of an adequate quality, we shall read through our code, and attempt to identify code that doesn't meet our standard (specified below) [\[10\]](#).

1. Legible - The code (the code itself, not comments) should clearly state the intent. If the reader can't make sense of the code, than all other efforts are doomed to frustration - if not outright failure.
2. Testable - The code should be organized in a way that facilitates unit testing. That supports all subsequent efforts (refactoring for modification, correction of defects, revision due to changed specs, etc.)
3. Flexible - Dependencies, both on other code in the code base and arbitrary implementation choices, should be minimised. Hard-coded assumptions about data size, concrete classes or data structures, etc. make the code more brittle, and therefore harder to reuse or adapt.
4. Compliant - The code should comply with its requirements, functional and otherwise. (I don't state this as "correct" because the discussion about whether the requirements themselves were the "right" requirements is about the process or the environment, not about the code.)
5. Economical - The code should make reasonable use of system resources - memory, CPU, etc. (I don't state this as "efficient" because that word is too often misused, by limiting it to a single aspect, such as speed. Economy is simply about return on investment, and requires thought about all the resources being invested and all the measures of return.)

Test Summary report

Test Summary Report [\[6\]](#), [\[7\]](#) is an important deliverable which is prepared after testing is completed. The objective of this document is to explain various details and activities about the testing performed for the project, to the respective stakeholders in this case our client Matt Morgan.

The report has 8 steps that I have identified:

1. Purpose of the document

- a. Short description about the objective of preparing the document
- 2. Application Overview**
 - a. Brief description about the application tested
- 3. Testing Scope**
 - a. This section explains about the functions/modules in scope & out of scope for testing; Any items which are not tested due to any constraints/dependencies/restrictions
 - i. In Scope - features we decide to test
 - ii. Out of Scope - features not to test
 - iii. Items not tested
- 4. Metrics**
 - a. Metrics will help to understand the test execution results
 - i. No. of test cases planned vs. executed
 - ii. No. of test cases passed/failed
- 5. Types of testing performed**
 - a. Smoke Testing - This testing was done whenever a build is received for testing to make sure the major functionality are working fine,
 - b. System Integration Testing - This is the testing performed on the Application under test, to verify the entire application works as per the requirements.
 - c. Regression Testing -
 - i. Regression testing was performed each time a new build is deployed for testing which contains defect fixes and new enhancements, if any.
 - ii. Regression Testing is being done on the entire application and not just the new functionality and Defect fixes.
 - iii. This testing ensures that existing functionality works fine after defect fix and new enhancements are added to the existing application.
 - iv. Test cases for new functionality are added to the existing test cases and executed
- 6. Test Environment & Tools**
 - a. Details on Test Environment in which the Testing is carried out. Server, Database, Application URL etc.
- 7. Exit Criteria**
 - a. Exit Criteria is defined as a completion of Testing by fulfilling certain conditions like
 - i. All planned test cases are executed;
 - ii. All Critical defects are Closed etc.
- 8. Conclusion/Sign Off**
 - a. Green signal for the application to 'Go Live' or not, after the Exit Criteria was met.

Work Plan

So far we have been using electronic tools to track tasks, and analyse our workflow. We have also been having weekly group meetings, where we have been identifying individual progress made on tasks, as well as communication online. These informative group meetings also allow us to formally allocate tasks to team members, and discuss any problems we are having with completing a task. In this section we state the tools we have used to track our progress through these tasks, and our success with them.

Risk Analysis

In our presentation we described the risks and problems we expected to encounter, and procured methods of recovery in such a case. To further protect ourselves from future problems we create an assumptions form (depicted below); we submitted potential problems we expected to not run into - if we did however - we had specified a solution. This form has been used through the analysis, and design stages of our project; we plan to continue using this in the next stages of development.

Category	Assumption	Responsibility	Status	Actions
Analysis	The client will not change their requirements.	Project	Closed	Identify all the project requirements.
Analysis	The team will produce an adequately in-depth analysis before beginning to design software.	Project	Closed	Identify work required, then complete said work.
Design	The client will not change their requirements.	Project	Open	Negotiate system requirements with client, alter project scheduling to accommodate changes.
Design	User information we want will be available.	Project	Open	Build in contingency to allow algorithm to work with the information missing.
Design	Client doesn't change the requirements, after the requirements presentation.	Project	Closed	Regress back to analysis stage and research into solution again, based on new changes.
Implementation	Team have enough programming experience to tackle any programming related issues.	Project	Open	Allocate jobs to certain coders, based on the difficulty of the task, and their experience.
Design	The design document shall be completed by specified deadline.	Project	Open	Allocate more time to working on the project.
Design	The design produced shall meet our client's expectations.	Project	Open	Negotiate necessary changes, then make changes to design document.

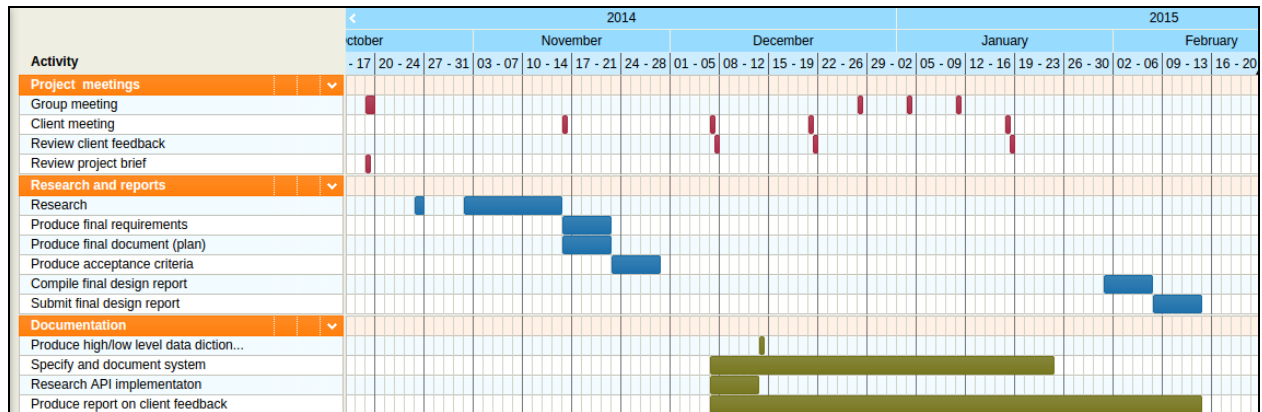
Gantt chart

The image below is a screenshot of our Gantt chart, made with an online tool [\[4\]](#), for our plan, up to the end of the design phase - as included in our presentation in week 7 of the autumn semester. While presenting this chart we failed to cover why our design does not include the respective members allocated to certain job roles. This was due to a couple factors, namely: group size, and conformity with RASCI chart.

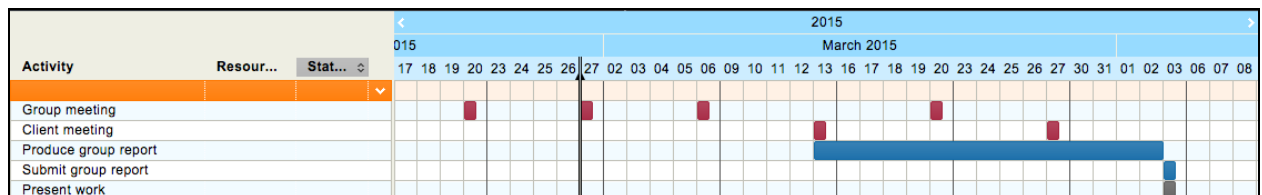
We have been using our own version of a RASCI chart to control task allocation, the chart identifies the tasks specified in the Gantt chart, and the main sub-tasks involved in completing a task - if team roles change.

Since our group is so small we find it to be quite easy to communicate, and manage job allocations. Thus the reason we needed a Gantt chart was less to control job roles, and more for identifying how much/what work we should have done by a given day.

Our adaption of these systems have worked for our group, providing the necessary flexibility for us to complete this project, while balancing our other work and commitments nicely.



Since last semester we have updated our Gantt chart to identify the timescales required to complete tasks we have to do during the implementation phase, testing, and evaluation stages. Once we have had our first client meeting, we shall make any necessary abrasions, and begin to split implementation tasks - these tasks shall be modularised, and specified in our RASCI chart.



RASCI chart

The image below is a screenshot of our RASCI chart; all items under the black horizontal rule are tasks that are part of the design process - these would come under the “Documentation” of the Gantt chart. This spreadsheet was hosted online so we could all access it, and find our job roles to see if there was any work we had to complete or could get ahead on.

In our RASCI chart we specifically do not use I's (informed), this was explained in our presentation, but for clarity we shall cover it again here. In the analysis and design stages the whole team needed to be informed. Due to our decentralised control structure information must be free to pass through the group to help. We plan to continue with this style of approach in the next stages of the project, as the clarity within the team was helpful in understanding what we were trying achieve with this project.

Tasks/Team Members	Abdu	Andrej	Ben	Karl	Rhian	Rob	Rob(o)	Sam	Zain	Project
Create Presentation	S	S	AR	S	S	S	S	S	S	
Review Client Feedback										AR
Research API		S	S	AR			S			
High Level Data Dictionary	S						AR			
Implement Dev. Model									AR	
Produce Presentation Report	AR	S	S	S	S	AR	S	S	S	
Develop Use Cases					S	S		S	AR	
Outline & Detail Use Cases								AR	S	
Produce Use Case Diagrams					AR					
Specify problem solving method	AR						S			
Identify Hardware		AR								
Identify Programming Language(s)				AR						
Low Level Data Dictionary			AR	S						
Produce DFDs					S	AR				
Class Diagrams				AR			S			
State Machine Diagrams		AR								
API Prototyping		S	AR	S						
Algorithm Prototyping		AR					S			
Design Dataframes							AR			
Design Wireframes								S	AR	
Produce Non-Functional UI design					AR					
Testing					S			S	AR	
Report Quality Control					S	S				

Implementation tasks

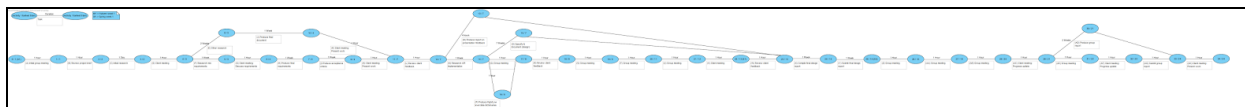
In the implementation stage the task we shall be attempting to complete is producing the required system in accordance with this document; to do this we shall split down tasks. Earlier in this document we identified that we could group our programmers into four roles: social media API coders, data transformers, algorithm creators, and front end developers. Once we have identified the people who best fit to each of these roles, we can begin to delegate tasks; these tasks shall be to build parts of/entire classes - identified in the class diagram above.

Critical Path Analysis (CPA)

We have been keeping our CPA diagram up to date, to help us keep track of our progress through the project. The breadth of the information identified on the diagram has been extended to spread to the end of the project, with all foreseeable critical steps appended. Below you may find a diagram of the CPA, and that same diagram in a tabular form.

Diagram

Due to the inappropriate shape of the CPA diagram we have included the latest version in the appended zip file - under the name 'CPA_Diagram_(5).png'.



Tabulated CPA

(Start date: 7th October 2014)

(Group report on approach towards solution due: 12th February 2015)

Task #.	Task	Earliest Start (Week)	Length	Type (Parallel / Sequential)	Dependent on...
A.	Initial group meeting	1 (A1)	1 hour	Sequential	

B.	Review project brief	2	1 hour	Sequential	A
C.	Initial research	2	1 day	Sequential	B
D.	Client meeting: Establish first	3	1 hour	Sequential	C
E.	Research into: - algorithm - hardware - business plan - risk analysis - legal issues - software design methodology - team structures	3	2 weeks	Sequential	D
F.	Research into requirements	3	1 week	Sequential	D
G.	Client meeting: Discuss requirements	4	1 hour	Sequential	F
H.	Produce final requirements	4	1 week	Sequential	G
I.	Produce acceptance criteria	5	1 week	Sequential	H
J.	Produce final document: - algorithm - hardware - business plan - risk analysis - legal issues - software design methodology - team structures	4	1 weeks	Sequential	E
K.	Client meeting: Present work - main contents (for report) - system requirements - acceptance criteria - legal, social, ethical & professional issues	7	1 hour	Sequential	H,I,J

	- work plan				
L.	Review client feedback	7	1 hour	Sequential	K
M.	Produce report on client feedback from requirements presentation	7	4 hours (required at end/in 16 weeks)	Parallel	L
N.	Research API Implementation	7	1 week	Sequential	L
O.	Specify and document: - method of solving problem - use cases - inter-relationships within system - hardware - programming language(s) - algorithms (prototyping when appropriate) - data flow diagrams - class diagrams	7	7 weeks	Sequential	M
P.	Produce High/Low level Data dictionaries	8	1 hour	Sequential	N
Q.	Group meeting	9	1 hour	Sequential	L
R.	Review client feedback	9 (A9)	1 hour	Sequential	Q
S.	Group meeting	10	1 hour	Sequential	R
T.	Group meeting	11	1 hour	Sequential	S
U.	Group meeting	12	1 hour	Sequential	T
V.	Client meeting: progress update	13 (S1)	1 hour	Sequential	U
W.	Review client feedback	13	1 hour	Sequential	V

X.	Compile final design report	15	1 Week	Sequential	O,P
Y.	Submit final design report	16 (S3)	1 week	Sequential	X
Z.	Group meeting - Review client feedback	18	1 hour	Sequential	Y
AA.	Group meeting - Identify implementation relevant skills - Delegate tasks	19	1 hour	Sequential	Z
AB.	Group meeting - Discuss problems found in tasks	20	1 hour	Sequential	AA
AC.	Client meeting: progress update - Present progress (implemented code) - Discuss group report/presentation	21 (S8)	1 hour	Sequential	AB
AD.	Produce group report: - Software functionalities - Match solution components to problem - Compare solution to proposed design - Compare methodology used to proposed methodology (algorithm, agile work, etc.)	21	3 weeks	Parallel	AC

	<ul style="list-style-type: none"> - Prove understanding of problem - Analysis of performance - Identify evidence of teamwork, and integration process - Justify all the above 				
AE.	Group meeting - Identify tasks yet to complete - Re-delegate tasks	22	1 hour	Sequential	AC
AF.	Client meeting: progress update - Present progress (newly implemented code)	23 (S10)	1 hour	Sequential	AE
AG	Submit group report	24	1 hour	Sequential	AD,AE
AH	Client meeting: present work	24 (S11)	1 hour	Sequential	AG

References

Research in requirements presentation

The following journals/papers are included have been appended to the zip file; the titles of the papers match up with the names of the PDF documents.

[] McPherson, M & Smith-Lovin, L & Cook, J M 2001, 'Birds of a Feather: Homophily in Social Networks', *Annu. Rev. Sociol.*, Vol. 27, pp. 415-44

[] Mark S. Granovetter, 1973, 'The Strength of Weak Ties', *American Journal of Sociology*, Vol. 78, Issue 6, pp. 1360-80

[] Gilbert, E & Karahalios, K 2009, 'Predicting Tie Strength With Social Media', In *Proc. of CHI*

[] Robert A. Hanneman & Mark Riddle, 2005, 'Introduction to social network methods', <http://faculty.ucr.edu/~hanneman/nettext/C3_Graphs.html>

Design report

- [1] Statista, 2014, 'Facebook: figures of monthly active users 2014',
<<http://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/>>
- [2] Statista, 2014, 'Google Plus: monthly active users worldwide 2013',
<<http://www.statista.com/statistics/283870/google-plus-monthly-active-users-worldwide/>>
- [3] Statista, 2014, 'Twitter: number of monthly active users 2014',
<<http://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/>>
- [4] Tom's Planner, 2015, 'Tom's Planner: Online Gantt Chart - Project Planning software',
<<http://www.tomsplanner.com/>>
- [5] Software Testing fundamentals, 2011, 'Test Case | Software Testing Fundamentals',
<<http://softwaretestingfundamentals.com/test-case/>>
- [6] Software Testing Help, 2015, 'A Simple 12 Steps Guide to Write an Effective Test Summary Report',
<<http://www.softwaretestinghelp.com/test-summary-report-template-download-sample/>>
- [7] Software Testing Help, 2015, 'How to Report Test Execution Smartly',
<<http://www.softwaretestinghelp.com/test-execution-report/>>
- [8] Webberley, W 2014, 'Generating Designs and Prototyping', Cardiff University
- [9] AgileMethodology.org, 2014, 'What is the Agile Software Development Methodology',
<<http://agilemethodology.org/>>
- [10] Stack Overflow, 2015, 'How do we define Code Quality?',
<<http://stackoverflow.com/questions/405243/how-do-we-define-code-quality>>

Appended Zip File Information

Note - The prototypes do not include all the code necessary to run - only the code we produced is included - demonstrations of the provided prototypes may be requested in a prearranged meeting.

Zip Structure:

- Work.zip
 - Prototypes
 - Facebook_API
 - matlabv3.html
 - logindetect.php
 - autoload
 - LIWC
 - liwc.php
 - TF-IDF
 - tf_idf.php
 - CPA_Diagram_(5).png
 - Predicting_Tie_Strength_With_Social_Media.pdf
 - Birds_Of_A_Feather_(Homophily_in_Social_Networks)

- The_Strength_Of_Weak_Ties